On the Elastic Optimisation of Cloud IaaS Environments

Chatziprimou, Kleopatra

Awarding institution:
King's College London

On The Elastic Optimisation of Cloud IaaS environments

A Dissertation submitted to the faculty of King's College London in Candidacy for the Degree of Doctor of Philosophy

by

Kleopatra Chatziprimou

Department of Informatics

Advisers: Kevin Lano, Steffen Zschaler

March 7, 2016

Abstract

Elasticity refers to the auto-scaling ability of clouds towards optimally matching their resources to actual demand conditions. An important problem facing the infrastructure and service providers is how to optimise their resource configurations online, to elastically serve time-varying demands.

Most scaling methodologies provide resource reconfiguration decisions to maintain quality properties under environment changes. However, issues related to the timeliness of such reconfiguration decisions are often neglected. A trade-off between the optimality of the reconfiguration solutions and the time cost to obtain these solutions is evident in the current literature. Highly accurate algorithms require a lot of data and time to execute, while more simplistic models may be fast to converge but provide poor quality solutions.

In this thesis, we present a methodology for online optimisation of cloud configurations. Our motive is to balance the optimality versus timeliness trade-off in dynamic configurations management. We first employ a search-based approach to extract near-optimal configurations considering mutually conflicting performance and business quality attributes. Towards reducing the burden of time-consuming fitness evaluations of the configurations' quality during search-based optimisation, we develop surrogate models to predict the configurations' quality based on history observations. We evaluate our technique using CloudSim-based cloud simulation. Our experimental results show that the proposed methodology can produce high quality configurations with lead time of seconds and prediction error within 6%.

Publications

During my doctoral studies I authored and published the following papers:

1. Kleopatra Chatziprimou, Kevin Lano and Steffen Zschaler : **Resources Provisioning in the Cloud**. In Proceedings of the First International Conference on Information and Communication Technologies for Sustainability (ICT4S), Feb. 14-16, ETH Zurich, Switzerland, 2013 (extended abstract).

2. Kleopatra Chatziprimou, Kevin Lano and Steffen Zschaler :**Towards a Metamodel of the Cloud Computing Resource Landscape**. In Proceedings of 1st International Conference on Model-Driven Engineering and Software Development (MODELSWARD), Feb. 19-21, Barcelona, Spain, 2013, p. 111-116.

3. Kleopatra Chatziprimou, Kevin Lano and Steffen Zschaler : **Runtime Infrastructure Optimisation in Cloud IaaS Structures**. In Proceedings of IEEE CloudCom 2013, December 2-5, Bristol UK, p. 687-692. Best Student Paper Award.

4. Kleopatra Chatziprimou, Kevin Lano and Steffen Zschaler : **Surrogate-Assisted Online Optimisation of Cloud IaaS Configurations**. In Proceedings of IEEE CloudCom 2014, December 15-18, Singapore. Acceptance rate 18%.

Acknowledgements

First of all I would like express my sincere gratitude to my supervisors Dr Kevin Lano and Dr Steffen Zschaler for their excellent supervision without which this work would have been impossible. I am grateful for my supervisors' patience, encouragement and advice the last three years. Particular thanks to them for giving me the opportunities to attend a number of international conferences and summer schools. I will forever be indebted to them for all they have given me. I would also like to thank my assessors, Professor Mark Harman and Dr Irakli Paraskaki. I am very grateful for their feedback and encouragement.

Throughout these years I had also the opportunity to collaborate with a number of people who critically contributed to this work. In particular I would like to acknowledge the help of all RELATE fellows and scientific advisers for the opinions they gave on my work and the stimulating discussions we had over research, that helped me progress my efforts and overcome limitations. In conclusion I would like to acknowledge the financial contribution of the EU RELATE ITN project and its partners for making this thesis possible. Finally, I am immensely grateful to my family; Efstathia, Theodosis and Thomas, for their support, that they demonstrate in so many ways.

To my grandmother, Eudokia.

Contents

Abstract iii

publications iv

Acknowledgements v

List of Tables xi

List of Figures xiii

1 Introduction **1**

1.1 Motivation 1

1.1.1 The Cloud Computing Paradigm 2

1.2 Problem Statement 4

1.3 Research Hypothesis 8

1.4 Thesis Contributions 10

1.4.1 Primary Contributions 10

1.4.2 Secondary Contributions 11

1.5 Thesis Structure 11

2 Literature Review **13**

2.1 Background 14

2.1.1 Virtualisation 14

2.1.2 SBSE Artefacts 17

2.1.3 Statistical Regression Foundations . . . 25
2.1.4 Regression Analysis . . . 26
2.2 Reactive Methods . . . 44
2.3 Proactive Methods . . . 58
2.4 Conclusions . . . 66

3 Research Scope **71**
3.1 Over-provisioned and Under-provisioned Datacenters. . . . 72
3.2 An Industrial Workload Study . . . 74
3.3 Use case Scenario . . . 82
3.4 Problem Statement . . . 85
3.5 Problem Characteristics Summary . . . 86
3.6 Use-case Simulation Settings . . . 88
3.7 Summary . . . 91

4 Formalisation of the Cloud Design Space **93**
4.1 Towards an Automated Configurations' Improvement Method . . . 93
4.2 The Cloud Metamodel . . . 95
4.2.1 Cloud Components . . . 97
4.2.2 Cloud Variability . . . 99
4.3 Summary . . . 102

5 Search-Based Optimisation of Cloud Configurations **105**
5.1 Reformulating Cloud Configuration Optimisation as SBSE Problem . 106
5.1.1 Initialisation . . . 110
5.1.2 Genetic Operators . . . 111
5.1.3 Genotype-to-Phenotype Mapping . . . 119
5.1.4 Fitness Functions . . . 120
5.1.5 Summary . . . 121

6 Fitness Function Approximation **124**

6.1 Approximate Fitness Functions 125

6.2 Specifying Surrogates for Fitness Approximation 127

6.2.1 Selection of Relevant Variables 127

6.2.2 Data Collection 131

6.2.3 Surrogate Models Description 139

6.2.4 Surrogate Models Selection 145

6.3 Summary 147

7 Evaluation Study **149**

7.1 Research Questions 150

7.1.1 Metrics Used 153

7.1.2 RQ1 155

7.1.3 RQ2 160

7.1.4 RQ3 168

7.1.5 RQ4 171

7.1.6 RQ5 172

7.2 Conclusions 174

7.3 Threats to Validity 175

8 Conclusions **182**

8.1 Thesis Contributions 184

8.2 Assumptions and Limitations 188

8.3 Open Research 189

A Appendix **192**

A.1 Covariance and Correlation 192

A.2 Statistical Testing 193

A.2.1 p-Values 193

A.2.2 The Wilcoxon Test 193
A.2.3 Cohen Effect Size Test 194

Bibliography **195**

List of Tables

2.1 Common kernel choices. 38

3.1 CAS PIA transaction types and frequencies. 76

5.1 Formalised Degrees of Freedom (DoF) in Cloud Resource Configurations. 107

5.2 Design of used genes in the chosen representation. 108

5.3 Mutation rates. 119

6.1 Potential predictor variables. The predictor values will vary among different cloud configurations and explain variations in the metrics of interest Q_{RT} and Q_E accordingly. 130

6.2 Reduced predictor variables. 133

6.3 Final predictor variables. 138

6.4 Training data ranges. 139

6.5 Regression Output for $\hat{Q_{RT}}$. 140

6.6 Regression Output for $\hat{Q_E}$. 140

6.7 Examined Model Parameters 143

6.8 146

7.1 Quality indicator values of random search and NSGA-II using the expensive fitness function. 160

7.2 Interpolation configuration for the predictors. 162
7.3 Extrapolation configuration for the predictors. 163
7.4 Hypervolumes achieved for extrapolated workloads. 166
7.5 Comparison of quality indicators and execution times between expensive and surrogate-based optimisation with NSGA-II. 168
7.6 Build times of surrogates. 172
7.7 Comparison of quality indicators and execution times between expensive and surrogate-based optimisation with ϵ-MOEA. 174
7.8 The used parameter values for the BoT characteristics based on [105]. N, P and W stand for the Normal, Pareto and Weibull distributions. 180

List of Figures

1.1 The cloud stack architecture [130] 4

1.2 Overview of our cloud configuration optimisation approach. 9

2.1 Abstract illustration of the representation and fitness function artefacts. 19

2.2 The fitness function assignes to each genotype candidate in the decision space an objective value in the objective space. 19

2.3 Illustration of single-point crossover (left) and two-point crossover (right). 23

2.4 A schematic illustration of the iterative nature of regression processes. Assumptions are initially made about the models and tested using a series of test statistics and graphs. Based on this testing feedback, assumptions can be refined or changed continously till the desired results are achieved. 26

2.5 From Left to right: high bias, balanced bias-variance trade-off, high variance. 28

2.6 Linear least squares fitting [92]. We seek the linear function of X that minimizes the sum of squared residuals from Y. 31

2.7 Classification and Regression Trees (CART) with 5 partitions [92]. . . 33

2.8 Piecewise linear functions with knot $t = 0.5$ [92]. 36

2.9 Slack variables [45]. 38

2.10 Left: samples of prior distributions without observed data points. Right: samples of posterior distribution after 2 data points have been observed. [165]. 39
2.11 Data split example. 40
2.12 10-fold cross-validation. 41
2.13 Bias and variance as a function of model complexity [71]. 44

3.1 The effect of elastic adaptation [113]. 72
3.2 The CAS PIA user interface with mock data. 75
3.3 ACF for transaction arrivals. 78
3.4 PACF for transaction arrivals. 78
3.5 Comparison of the actual and predicted observations of the transaction arrivals time series. 78
3.6 Histogram of CAS CPU utilisation. We observe that mainly CPU usage falls within 20%. 79
3.7 CPU utilisation time series. 80
3.8 ACF for the CPU time series. 80
3.9 Elastic architecture vision based on [47]. 82
3.10 Cloud configuration example. 83
3.11 Reconfiguration to match increasing workload. 84
3.12 Energy-performance trade-off in a cloud of 100 PMs serving Web services' load [33]. 87
3.13 The CloudSim layered architecture [36]. 89

3.14 BoT example. The Y axis shows different users and the X axis shows the time. We observe that user U_1 has submitted three BoTs; G_1, G_2 and G_3 while the user U_2 has submitted a single BoT G_4. The BoTs G_2 and G_3 contain one task while the BoT G_1 contains two tasks and the BoT G_3 contains three. The individual tasks within BoTs may have different staring times as well as different runtimes [105]. 90

4.1 Cloud System Metamodel. Each box is a meta-class and the lines represent references between meta-classes. References and attributes of a meta-class are known as meta-features. The black diamonds on the relations represent composition, meaning e.g. in this Figure that Wrokload, Configuration and Component can only exist if they are contained in the Cloud. The asterisks on a reference specifies the multiplicity of this reference. Fianlly the white triangular arrowhead represents inheritance. In this exmple we see that both Workload and Cloud inherit from NamedElement. 96

4.2 Cloud Infrastructure. 98

4.3 Cloud Variability. 100

4.4 A configuration model instance example. 103

5.1 The genotype of our representation 106

5.2 Overview of VM and PM chromosomes. Symbols 1 and + indicate that the elements occur exactly once and more than once respectively. 109

5.3 Crossover operator example. 112

5.4 Mutation operator example. The M-VD operator removes a VM chromosome from its PM host. 115

5.5 Mutation operators priorities. 117

5.6 The genotype-to-phenotype process. The cloud genotypes are assigned mutation and crossover operators and checked for structural cosistency. Each genotype is mapped to a simulated cloud datacenter using CloudSim. The output of the simulation is the score (Q_{RT} and Q_E) for each candidate. 120
5.7 Optimisation process. 122

6.1 Scatter plots of predictors cores and PMs versus the responses energy Q_E and response time Q_{RT} 131
6.2 Scatter plots of predictors CPU and RAM allocation versus the responses energy Q_E and response time Q_{RT} 132
6.3 Scatter plots of predictor processor speed versus the responses energy Q_E and response time Q_{RT} 133

6.4 Pair-wise scatter plots and correlation measures between the response variable Q_{RT} and the predictors in the recorded dataset. The numerical values inside the grid boxes, highlighted under the three red stars notion, show the correlation values. The diagonal boxes of the grid show the data distributions of the studied variables. From left to right in the diagonal we see the data distribution of the dependent variable; i.e., *response time* and predictors i.e., *load*, *PMs*, *VMS*, *allocated CPU* and *resource size*. All other boxes in the grid depict pair-wise relationships between the response and the predictors and among the predictors themselfs. The Y axis corresponds to the variable described in the diagonal box of the same **row** and the X axis corresponds to the variable described in the diagonal box of the same **column**. The grid boxes below the diagonal show the graphical relationships between the corresponding Y and X variables. The grid boxes above the diagonal show the correlation values between the corresponding Y and X variables. For example in the first box of the second from the top row show the function between *load* (Y axis) and *response time* (X axis). In the second box of the top row the correlation value between *load* and *response time* is shown. 134

6.5 Pair-wise scatter plots and correlation measures between the response variable Q_E and predictors i.e., *load*, *PMs*, *VMS*, *allocated CPU* and *resource size* in the recorded dataset. 135

6.6 Pair-wise scatter plots and correlation measures between the response Q_{RT} and predictors i.e., *load*, *PMs*, *VMS*, *allocated CPU* and *resource size* in the transformed dataset. 136

6.7 Pair-wise scatter plots and correlation measures between the response Q_E and predictors i.e., *load*, *PMs*, *VMS*, *allocated CPU* and *resource size* in the transformed dataset. 137
6.8 Diagnostic checks for Response Time. 141
6.9 Diagnostic checks for Energy. 141
6.10 3-dimensional scatter view of the recorded transformed data. 144
6.11 MARS models selection. 144
6.12 Identifying surrogates with sufficient predictor subsets [64]. 145
6.13 Identifying surrogates with sufficient predictor subsets [64]. 146
6.14 Prediction accuracy for the surrogates. 147
6.15 Model construction overview. 148

7.1 Example of hypervolume for a minimisation problem with referenco point z [124]. 154
7.2 The NSGA-II concepts of dominance ranking and crowding distance. 156
7.3 Hypervolume evolution through optimisation with NSGA-II and random search. 159
7.4 NSGA-II and random search Paretos in comparison to the initial population. NSGA-II achieves an improvement up to 79% for response time and 88% for energy consumption w.r.t the problem input. . . . 160
7.5 Extrapolation for smaller workloads. 164
7.6 Extrapolation for larger workloads. 165
7.7 Changes in MAPE as the size of current dataset increases. 167
7.8 Surrogate I_H overview. 169
7.9 Monitored metrics; energy consumption, response times and utilisation ratios[42]. 177
7.10 SPECpower benchmark real data on server power consumption. . . . 178

8.1 Cost of a single VM migration for a three-tier web application. Three deifferent workloads of 100, 400 and 800 concurrent users are shown. [114] 189

Chapter 1

Introduction

1.1 Motivation

MASSIVE data growth, challenging economic conditions, and the physical limitations of power consumption and space availability have been exerting pressure on today's enterprises [113]. Common datacenter design malpractices such as overprovisioning fixed resource capacities [118, 129, 13, 25] to anticipate peak workloads, have resulted over the years in low server utilisation, uncontrolled expenditures, and difficulties to keep up with service's demand [123, 184]. Enterprise workloads can change in terms of both intensity and requests mix[77]. To manage the changing application requirements, different amounts and distributions of resources should be allocated to applications, rather than static at all times, dedicated hardware capacities[77]. Therefore, investigating new architectures with potential to deliver more business value to providers has become an important objective for the industry.

The cloud computing [131, 147] paradigm offers a solution to the above concerns. Computing facilities (e.g., remote storage) accessed via APIs, are provided by a cloud provider to a cloud client. Often a client may be a third party company, who will use the computing facilities to provide a service to their users [149]. For example,

Amazon is a cloud provider supplying a storage API to Dropbox, who use the API to provide file synchronisation service to domestic and commercial users [97].

1.1.1 The Cloud Computing Paradigm

Several areas of research have converged to the single concept of cloud computing, defined by the National Institute of Standards and Technology (NIST) [147] as:

> *"A model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."*

The existing technologies, that have been merged into the notion of cloud are:

- *Grid computing:* A distributed computing paradigm that coordinates networked resources to achieve a common computational objective [72].
- *Utility computing:* A model for providing resources on-demand and charging customers based on actual usage rather than a flat rate [170].
- *Virtualisation:* A technology that abstracts away the details of physical hardware providing the capability of dynamically (re)assigning virtual resources to applications on-demand [177, 16].
- *Autonomic computing:* An architectural model focusing on the engineering of so-called self-aware IT systems and services [110]. The vision of self-awareness describes systems that are *(i) self-reflective* i.e. aware of their software architecture, hardware infrastructure and operational goals (e.g., QoS requirements, cost- and energy-efficiency targets), *(ii) self-predictive* i.e. able to predict and anticipate possible performance problems, *(iii) self-adaptive* i.e., adapting as

the environment evolves in order to ensure that their operational goals are continuously met and *(iv) self-optimising* i.e. able to decide optimal actions to improve their execution.

The central technical attributes of the cloud paradigm are summarised in the concept of *elasticity*. Elasticity defines the ability of IT systems to automatically adapt to environment changes by provisioning (scale out) and de-provisioning (scale in) resources, such that at each time point the available resources match the current demand as closely as possible [95]. As a result users experience an effect of infinite computing resources available on-demand.

Cloud computing represents a shift towards new operational model of computing. Rather than viewing software and the hardware upon which it executes as fixed assets to be purchased and therefore subject to depreciation, the cloud model treats both software and hardware as rented commodities. This is a profound change, with potential to remove both the business issue of depreciation and technical concerns such as resource fragmentation, scalability and platform interoperability [87]. Cloud computing is often presented as a layered architecture as shown in figure 1.1.

The infrastructure layer is typically composed of hardware arranged in racks and clusters of racks. The racks are physically located in series of geographically distributed datacenters. Hardware servers host and manipulate multiple virtual machines using virtualisation [169] . Infrastructure-as-a-Service(IaaS) provide a management front-end API for a set or pool of resources in order to offer to higher level services or third parties, demand based scalability, fail-over and operating system hosting [130]. IaaS examples are MapReduce [56] and GoogleFS [79]. The platform layer entails programming and execution environment services. For example, customers provide code to Google App Engine [103] and Google automatically manages execution and scaling. The top layer contains applications running in the cloud, offered directly to the general public. Popular current applications are Microsoft Office

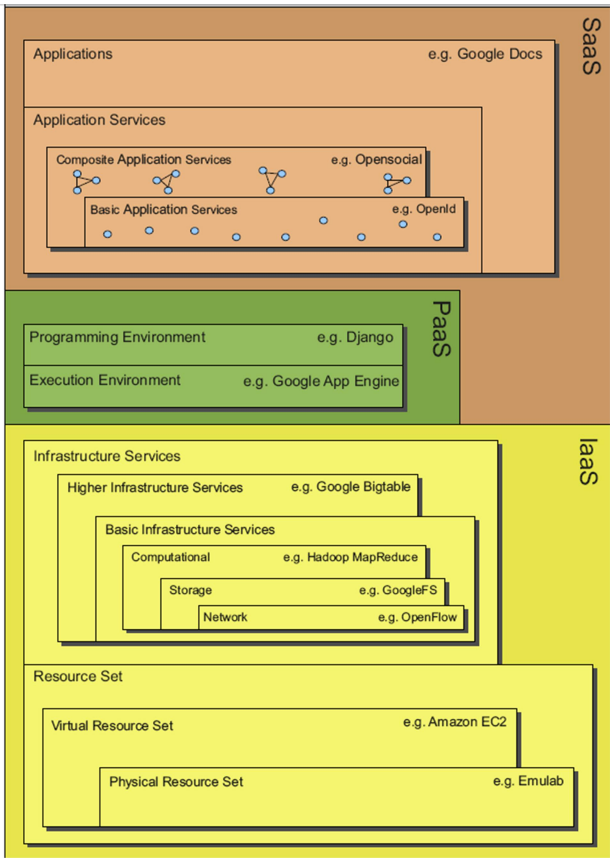


Figure 1.1: The cloud stack architecture [130]

.

Live [141] and Google Docs [102]. Work in modelling interactions among the cloud layers [149], reports complex patterns and strong inter-dependencies between layers. As an example, an end-user for executing a given task may use services from only a single IaaS, PaaS, or SaaS provider, or might use resources from multiple such providers at the same time. As a result rather than clearly distinguishable IaaS, PaaS, and SaaS service delivery models, a so-called *mixed delivery model* is gaining increasing popularity [180].

1.2 Problem Statement

In this study we focus on the IaaS cloud layer. An IaaS-level requirement that brings advantages to the whole cloud stack is the so called *elasticity*. Elasticity is defined

as the ability of cloud datacenters to add (scale-out) or remove (scale-in) resources at a fine grain such that at each point in time the available datacenter resources match the current demand as closely as possible [10, 95]. The *precision* of scaling is a critical elasticity attribute. If the resource requirements of an application are not fulfilled, the application can face increased response times, time-outs and failures. Therefore adequate resource provisioning is directly linked to the Quality of Service (QoS) i.e., measurable levels of quality attributes such as availability, reliability and performance.

Capacity planning methods are central for *provisioning resources* to applications and therefore guaranteeing continuous QoS in dynamic environments [117]. Capacity planning relies on resource partitioning and allocation towards ensuring predictable behaviour of applications. The goal is to identify cost-effective resource configurations to optimally cope with environment changes such as workload fluctuations at runtime. However, traditional datacenters require a lead time of weeks to scale resources [10] and therefore have resorted to fixed dedicated hardware to anticipate demand spikes.

This is because deciding how to optimally modify datacenter configurations is a hard task. First, an optimal configuration must exhibit a good trade-off between mutually conflicting non-functional goals such as applications' performance and configuration cost. Second, the space of all possible datacenter configurations is huge due to the number of the available configuration parameters such as CPU time, memory, and network bandwidth. Therefore, using a manual or brute force search for finding an optimal configuration is infeasible. Third, clouds are highly volatile; parameters as workload demand, resource availability and energy price might change at run-time. Such changes require the initially chosen datacenter configuration to be accordingly updated. Finally, the cost of the optimisation process itself makes runtime configuration adaptation complex because of the additional time and computational resources needed to find the improved cloud configurations. For example when the workload

is rapidly changing it may be better to suffer slight performance degradations than trigger expensive reconfiguration actions, whose costs may never be recouped before a new reconfiguration is needed.

The second critical attribute of elasticity is the *speed* of resources adaptation. Rapid elasticity is critical to eliminate over-provisioning without compromising the applications' performance under dynamically changing conditions. Unfortunately cloud providers have not yet solved the problem of finding optimal datacenter configurations at runtime, to adapt their systems to non-trivial, dynamic changes such as unexpected fluctuations, sudden spikes in the service demands, complexity in virtualised applications and heterogeneity of resources inside the clouds [77]. Existing commercial clouds such as Amazon EC2 rely on users to specify their own rules for scaling resources according to their needs [172]. Such rule-based approaches tend to be wasteful in terms of resource usage [69]. The current literature offers a lot of possible methods to reconfigure cloud datacenters, however a trade-off between the precision of the solutions and the cost as well as time to obtain these solutions is evident. Very accurate algorithms (such as Search-based Software Engineering, briefly described below) need a lot of data and time to execute [86, 74] while more simplistic algorithms (such as greedy heuristics) are very fast to execute but provide poor estimations [174, 66].

In this thesis, we aim to extend the current state-of-the art with a methodology to balance the trade-off between optimality and timeliness of the reconfiguration decisions, towards enabling elasticity in cloud datacenters. Our approach considers cloud services as black-box resource requests. This black-box model enables us to abstract from the internal services architecture and focus on the infrastructure provider's viewpoint, that is often services-agnostic due to legal boundaries among different organisations [149]. The main components of our proposed methodology are described below.

Search-based Software Engineering

Search-Based Software Engineering (SBSE) is an engineering approach that focuses on reformulating software engineering problems as optimisation problems and applying metaheuristic optimisation techniques to discovering (near) optimal solutions to the problem [89]. The interest in SBSE has been growing in recent years , with it being successfully applied to all aspects of the software development life-cycle [12, 68, 199]. To reformulate a software engineering problem as an optimisation one needs to perform two actions; (i) define a representation of the problems solution space that can be used within the metaheuristic technique and (ii) devise a fitness function to measure the quality of each candidate solution [89]. The output of the fitness function is used to guide the metaheuristic technique towards optimal solutions. In our approach, we will first exploit SBSE algorithms to search for near optimal cloud configurations.

Surrogate-based Optimisation

In SBSE algorithms it is often implicitly assumed that there exists an explicit fitness function formula to evaluate solutions [111]. However in the majority of real-world problems fitness evaluations are not straightforward because analytical formulas to associate solution candidates with quality metrics do not exist. In cloud environments particularly, applications may have complex software architectures and it is not always clear what are the relationships between system configurations, parameter settings, resource allocations, incoming workloads and the end-to-end application QoS [77].

As a result, fitness values are typically estimated using computation simulations or physical experiments. In practice such tactics become non-trivial as each fitness evaluation is highly time-consuming. Therefore the overall SBSE convergence becomes prohibitively costly. Surrogates can be used to alleviate the computational burden, providing an alternate and explicit expression for the fitness function, based on data describing the mapping between the design parameters and the quality of the design

[134]. Using the approximated fitness functions in the SBSE formulation is orders of magnitude cheaper to run, than the full analysis.

1.3 Research Hypothesis

The research hypothesis addressed in this thesis is as follows:

A generic surrogate-assisted search-based reformulation of the cloud configuration improvement problem, would enable the high-quality optimisation results achieved with SBSE to a wealth of existing research, to be applied at runtime.

Search-Based Software Engineering have been successfully applied to all aspects of the software engineering life-cycle [90, 6, 12]. More recently SBSE techniques are being applied to achieve near-optimal solutions for many of the problems posed by the migration of computation to cloud computing platform [74, 87]. However, SBSE complexity highly depends on the complexity of the fitness function. As in the datacenter configuration optimisation problem, there is no obvious analytical expression for estimating the quality metrics of interest for datacenter configurations, Highly time-consuming simulations are typically employed to evaluate configurations, leading to cost prohibitive optimisation processes, that require hours or even days to converge. In the following we briefly describe the approach adopted in this work, to achieve a balance between optimality of the configurations and the timeliness of convergence, for the cloud configuration optimisation problem.

Approach Overview

Our approach aims to provide an on-line optimisation framework to cloud providers, to adjust the configurations of cloud systems at runtime in order to maintain contin-

uous QoS despite possible environment disturbances. The QoS properties we study are the cloud applications' response time Q_{RT} and cloud datacenter total energy consumption cost Q_E. Our focus on this problem is to balance the trade-off between the optimisation output quality and the time needed for the optimiser to converge.

Figure 1.2 provides a high-level overview of our optimisation framework. We have assumed that a software cloud *monitoring* tool exists to provide notifications of changes in the cloud environment and therefore triggers to call the optimisation framework. Hence, input to our framework is a cloud configuration to be improved.

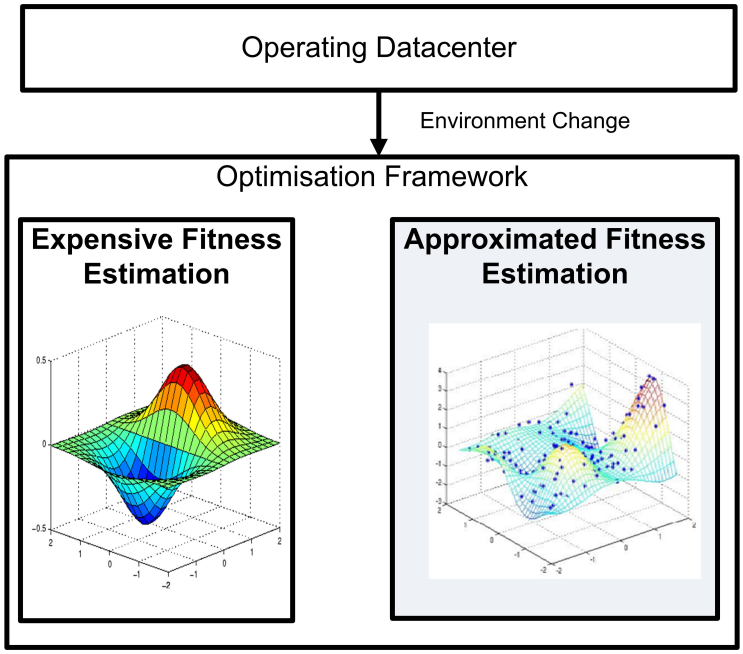


Figure 1.2: Overview of our cloud configuration optimisation approach.

Exact optimisation models need a lot of time to execute, while very simplistic models such as greedy heuristics are very fast to use but provide poor estimations. SBSE offers an efficient way to explore the vast cloud design space and explore optimal or near-optimal solutions by introducing smart operators, that balance the exploration of new areas of the space and exploitation of already explored good solution candidates. Thus, the **first step** in our method is the reformulation of the optimisation problem as SBSE, by encoding the input candidate in a genotype representation as discussed

in Section 5.1. The metrics Q_{RT} and Q_E act as fitness functions to measure the quality of individuals and differentiate better from worse candidates. However, in the cloud configuration optimisation problem, there is no obvious analytical expression to model the relationship between configurations' characteristics and the non-functional properties Q_{RT} and Q_E. We therefore resort to alternative methods i.e, simulation to measure the candidates' fitness. Yet, the use of simulation introduces delays in the execution of the meta-heuristic search and is not appropriate at runtime. The **second step** aims to bridge the gap between the non-functional properties (here; Q_{RT}, Q_E) and the cloud configurations, by approximating the unknown relation between them. As described in Section 6.2 a series of different surrogates is built for each fitness function. The meta-heuristic search replaces the use of the computationally expensive, simulation-based fitness functions with surrogates. Based on the above our initial research hypothesis (see Section 1.3) can be redefined as:

A generic surrogate-assisted search-based reformulation of the cloud configuration optimisation problem on the non-functional properties Q_{RT} and Q_E), would enable the high-quality optimisation results achieved with SBSE to a wealth of existing research, to be applied at runtime.

1.4 Thesis Contributions

The primary contributions made in this thesis are summarised below.

1.4.1 Primary Contributions

- The demonstration of feasibility to apply SBSE and achieve near-optimal trade-off cloud configurations considering the quality attributes of energy consumption and hosted services response time.

- The development of light-weight surrogate models by using statistical regression techniques, to make the SBSE optimisation process suitable at runtime.

- An empirical evaluation of our methodology on the extended CloudSim. Our findings show that our surrogate-assisted optimisation approach can guide the search to good trade-off cloud configurations with a lead time of seconds and prediction error within 6%.

1.4.2 Secondary Contributions

- The definition of a generic representation of models to describe cloud IaaS configurations that is amenable to a wide range of existing metaheuristic optimisation techniques.

- The extension of the current state-of-the-art cloud simulation platform CloudSim [36] with SBSE and surrogate features, integrating open-source tools such as the multi-objective optimisation platform MOEA [1] and the statistical environment R [2].

1.5 Thesis Structure

Chapter 2 presents a review of the related literature. First, we provide the necessary background information overview on virtualisation, search-based optimisation and statistical analysis to facilitate the understanding of our approach. Second we discuss related work in the field of cloud configurations optimisation.

Chapter 3 demonstrates our use-case scenario to motivate our research contribution and analyse the challenged we will address. We also formalise our problem statement and present our simulation environment settings.

[1] `http://www.moeaframework.org/`
[2] `http://www.r-project.org/`

Chapter 4 introduces our abstract representation of cloud IaaS configuration models. The proposed formalisation of the cloud infrastructure core structures and degrees of freedom serves to enforce design constraints and specify which valid changes may be implemented towards tuning the configuration quality attributes without affecting the functionality of the system.

Chapter 5 reformulates the problem of cloud configurations optimisation as SBSE problem. We describe the key components of our SBSE representation and use it to discover near-optimal solutions.

Chapter 6 addresses the timeliness concern for the problem of cloud configurations optimisation. First we describe the foundations of statistical regression and then apply these principles to devise a series of surrogates using past cloud historical data that describe the configuration components and their quality under different workloads.

Chapter 7 describes the evaluation process to validate our methodology towards balancing the optimality versus timeliness trade-off in the cloud optimisation problem. We first demonstrate that SBSE can be used to achieve near-optimal cloud configurations. Next we showcase the robust predictive ability of our devised surrogates and use them as alternate fitness functions under different environment conditions.

Chapter 8 concludes the work. It summarizes our methodology, lists the achieved scientific contributions, discusses the limitations of our solution, and illustrates the future research that can spring off our work.

Chapter 2

Literature Review

This thesis proposes a framework for elastic cloud configuration optimisation. In this chapter we review the existing literature in the area of resources provisioning and optimisation in cloud configurations. We have started our analysis studying traditional virtualised datacenters and providing a brief overview on virtualisation technology. These concepts are important for the foundation of cloud systems, as cloud virtual and physical resources are "pooled" together towards seamlessly serving multiple customers based on virtualization technology. Evolving the traditional virtualised environments, clouds employ a new *operational model* [209], that focuses on rapid elasticity, i.e. the ability of infrastructure to instantly scale up or down to optimally meet the customer demand while minimising capital and operating expenses. In this chapter we will track the evolution of resource provisioning and optimisation methods through the transition of virtualised datacenters to cloud datacenters. Our aim is to critically overview the literature and outline the gap in the area of elastic cloud optimisation.

The structure of this chapter is as follows; In Section 2.1 we provide overviews of virtualisation, SBSE and statistical regression that are necessary to understand this thesis. Then the critical review of existing literature in resource provisioning follows, classified under two different trends in Sections 2.2 and 2.3. The first category applies

traditional performance modelling techniques such as regression to estimate future application workloads and changes in the environment. As a result the system can anticipate environment changes and may scale in or out according to the predicted estimates. We will refer to this family of approaches as *proactive*. The second category adjust the cloud resources on demand, based solely on recent behaviour. As we will see such contributions mainly focus on the optimisation of configurations based on predefined single or multiple stakeholder objectives. We will refer to this family of approaches as *reactive*. Finally in Section 2.4 we present our conclusions and discuss the research gap.

2.1 Background

This section overviews the basic techniques over which the thesis develops; virtualisation, SBSE, and statistical regression.

2.1.1 Virtualisation

Virtualisation is a technology that abstracts away the details of physical hardware and provides virtualised resources to higher-level applications. A virtualised server is called a virtual machine (VM) [16, 168]. The most common methods in the development of VMs include *full system* virtualisation, where an entire hardware architecture is replicated virtually and *paravirtualisation*, where an operating system is modified so that it can be run concurrently with other paravirtualisation-compatible operating systems [168]. Virtualization allows applications to share the same physical server by creating multiple virtual machines in a manner such that each application can assume ownership of the virtual machine [192]. The use of VMs, enables many software environments to run concurrently on a single physical machine (PM) with high performance, providing better use of physical resources and isolating individual software

instances [120]. Software layers that control memory partitions and CPU scheduling from the host PM to the hosted VMs are called hypervisors. One of the most commonly used hypervisors is Xen [16]. Typically, a single application is considered to be executing in a single VM [16].

Virtualisation is a cloud computing foundation because it provides the capability of pooling computing resources and dynamically assigning and reassigning resources to applications on demand [209]. This flexible control of infrastructure resources is possible via managing the VMs e.g., by activating new VMs when demand surges and deactivating existing VMs when demand decreases. However, the main feature of virtualisation is *VM migration* [120], that allows to flexibly remap at runtime physical resources to virtual servers towards handling workload dynamics and hotspots. The concept of VM migration is considered the basic virtualisation feature towards improved performance, manageability and fault tolerance [196]. VM migration refers to the actual transition of a VM from a source PM to a destination PM in an isolated and fault tolerant fashion. The migration of a VM is typically triggered from a critical event such as e.g. CPU overload. The VM and its applications remain agnostic of the process. Migration can occur in a *stop-and-copy* or *live* mode [29]. The former moves a VM from a source host to a target by pausing the original VM, copying its memory contents and then reusing these to the destination VM. Migration with liveness constraints indicates that the VM continues to run while transferring its memory and local persistent state (i.e., the VMs file system). Live migration of VMs provide a significant benefit for virtual server mobility without disrupting service [137].

A popular method of implementing live migrations is the "iterative pre-copy" technique, where memory pages are iteratively pre-copied by the hypervisor from the source to the destination PM without ever stopping the migrating VM. Normally though, there is a set of pages that is modified so often that the VM must be stopped for a period of time, until this set is fully transferred to the destination [196].

It has been observed that live migration of VMs allows workload movement with a negligible application downtime. Nevertheless, the performance of a running application is likely to be negatively affected during the migration process due to the overhead caused by successive iterations of memory pre-copying [196]. The VM memory size has been found to mainly affect downtimes and network traffic [137]. During the pre-copying process extra CPU cycles are consumed on both the source and destination PMs. An extra amount of bandwidth is consumed as well, potentially affecting the responsiveness of web-based applications. In addition, when the VM resumes after migration, a slowdown is expected due to the cache warm-up at the destination. Downtimes affecting the applications' performance may vary due to the different applications' memory usage and access patters. Reported downtimes vary within the window of 60 ms to 3 seconds [196]. Finally VM migrations are associated with additional energy consumption costs. Previous studies have shown that the energy consumption is mainly affected by the VM data transmission rate [137].

Virtualisation Challenges in Modern Datacenters

Overall, virtualization technology allows cloud stakeholders to create multiple Virtual Machine (VMs) instances on a single physical server, and therefore improve the utilization of resources and increase the return of their investment. Reduction in energy consumption can be achieved by switching-off idle PMs and VMs to eliminate the idle power consumption, resulting from idle servers. Idle power consumption is not negligible and is estimated up to 60% of peak power consumption [146]. Moreover, by using live migration, the VMs can be dynamically consolidated to the minimal number of PMs according to their current resource requirements [22]. However efficient resource management in clouds is not trivial as modern service applications often experience highly variable workloads causing dynamic resource usage patterns [22]. If the resource requirements of an application are not fulfilled, the applica-

tion can face increased response times, time-outs or failures. The reconfiguration actions themselves pose additional computational costs and risks. For example VM migrations, the key reconfiguration action, impose latencies due to VM downtimes and additional energy consumption due to the additional CPU and memory required during the migration. Similarly the time to boot new hardware and virtual nodes might might hinder timely reconfiguration actions, that are critical especially when the workload is dynamically changing.

While aggressive consolidation of VMs offers a simple and straightforward resource management approach, it can also lead to performance degradations, particularly when applications experience increasing demands, that result in unexpected rise of resource usage. Additionally the total number of reconfiguration actions has to be controlled, towards minimising the performance penalties of these management actions. Dynamic scaling the infrastructure in response to workload or performance changes presents a key challenge for resources provisioning techniques [17], that has concerned the academic community the recent years and still appears to be an open problem. An effective scaling solution should allocate resources to optimise multiple objectives such as cost, performance and reliability[17]. However software quality attributes are often in mutual conflict: e.g., performance versus cost. In general improving one quality property can deteriorate another thus quality properties cannot be improved in isolation. Consequently, configurations that exhibit a good trade-off between multiple quality criteria must be achieved. The reconfiguration solution itself should also be scalable, i.e., capable of dealing with large, rapidly changing workloads and with the complex cloud resources design space.

2.1.2 SBSE Artefacts

In this Section we detail the key concepts of SBSE and metaheuristic search, that constitute the necessary background for our optimisation approach in Chapter 5. SBSE

is an approach that treats software engineering problems as optimisation problems. SBSE is based on the observation that it is often easier to check that a candidate solution solves a problem than it is to construct a solution to that problem. Indeed, solutions to some software engineering problems may be theoretically impossible or practically infeasible; SBSE techniques can help discover acceptable solutions to these problems [89]. The re-formulation of an engineering problem as a search-based optimisation problem requires two key ingredients according to SBSE practice:

- A problem *representation* choice.
- A *fitness function* definition.

The first SBSE artefact is an encoding of the problem domain. The problem *representation* defines a model of the studied system in a unique way, that is amenable to symbolic manipulation [89]. For several SBSE applications in the literature, a problem representation is simply a string of binary digits [145] or a floating point array [37]. This representation is then translated into the native format of the optimisation problem solution, to be manipulated by the *fitness function* [199]. *Cloud software optimisation* [74] for example, translates a representation of an integer vector to software components and reconfiguration rules, that define a cloud software application. This mapping from the search-amenable encoding representation to the real-world representation is so called *genotype-to-phenotype mapping* [171]. The representation is also known as *genotype*, while the *phenotype* correspond to the representation translation, used by the fitness function. Constituents of the genotype, such as bits or integers, are known as *genes* while *alleles* correspond to the specific values assigned to the genes. Genes can be grouped in larger structures that are called *chromosomes*. An important property of representations according to Rothlauf, is the *locality* [171]. The term is used to describe the effect of small genotype changes to the phenotype. In high-locality representations neighbouring genotypes correspond to neighbouring

phenotypes, while low locality relates to the opposite. High locality is important, as the search algorithm can be guided smoothly towards the optimal solution through small genotypic changes.

The fitness function is used to evaluate the solution candidates and measure how "close" they are to solving the given problem [199]. A fitness function assigns to each genotype a fitness score, that is used to differentiate better from worse solution candidates. More specifically, a fitness function $f : X \rightarrow Y$ assigns to each genotype candidate x_i in the *decision space* X an objective value y_i in the *objective space* Y, to evaluate the quality of each solution [215]. Figure 2.1 shows an abstract overview of the process and Figure 2.2 illustrates the fitness assignment process.

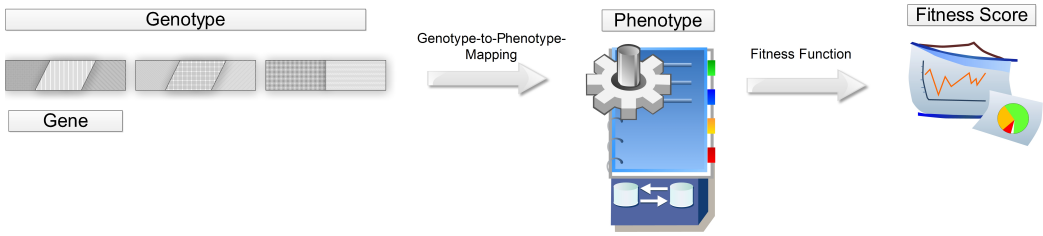


Figure 2.1: Abstract illustration of the representation and fitness function artefacts.

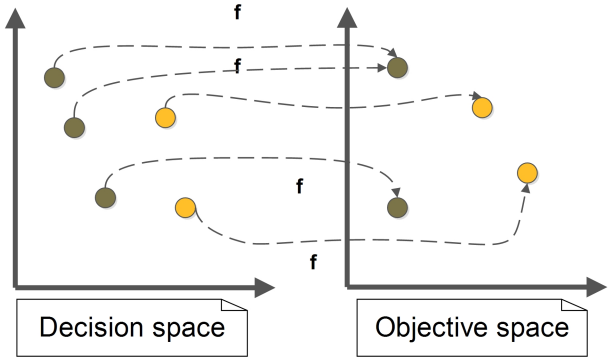


Figure 2.2: The fitness function assignes to each genotype candidate in the decision space an objective value in the objective space.

Metrics consist a natural choice for fitness functions, because they act as "projections" of qualities of interest in software systems [86]. If a metric does not meet the required quality standards, the software engineers will naturally take corrective

action. The one-to-one correspondence between metrics and fitness functions means that many metrics, such as lines of code or fault density, are a straightforward choice to guide the search for near optimal solutions [86]. This also allows metrics to move from being a passive assessment of a system or process to an active driver for improvement [90] Overall defining a suitable fitness function is an important task, critical to decide the outcome of an SBSE formulation. A well-defined fitness function is able to distinguish between subtly different solutions in such a way that it substantially increases the chance of guiding the search algorithm through the search space to some optimal solution [199].

Clarke et al. [51] specify four characteristics of software engineering problems that make them suitable for being reformulated as SBSE problems. These include:

- the problem has a *large* and *multi-modal* solutions search space. Such spaces typically arise when a number of competing constraints have to be balanced, to a arrive at an acceptable solution. Typically for such problems precise, analytic algorithms are infeasible, yet it is possible to determine the best between two candidate solutions.

- There are no known solutions that are efficient to compute.

- It is reasonably easy to determine the quality of a candidate. Fields of software engineering with a large number of readily accepted metrics, that can act directly as fitness functions , consist for example good SBSE practice areas.

- Generating solutions is computationally cheap. Commonly, search-based algorithms require many executions of the fitness function to converge to near-optimal areas of the search space. Therefore, the speed of execution of the fitness function is crucial.

After an appropriate representation and a fitness function have been defined for the problem, any *metaheuristic search* algorithm may be applied, to explore trade-offs in the problem design space.

Metaheuristic Search

Metaheuristic search algorithms are high-level optimisation algorithms that incrementally improve a given initial solution, until an optimal or near-optimal solution is identified [26]. They aim to efficiently explore the solution space of a problem i.e. the possibly infinite set of all possible problem solution candidates. As opposed to heuristic techniques which are designed to address specific problems, metaheuristics are not problem specific, and can be therefore applied to a wide range of problems [26]. Metaheuristics are usually non-deterministic and many of the metaheuristic approaches rely on probabilistic decisions made during the search for near-optimal solutions. They differ from random search though, as here randomness is not used blindly but in "an intelligent, biased form" [182]. The strength of metaheuristic algorithms is that they seek iteratively an optimum solution within a landscape that may be very complex and even discontinuous [51].

Metaheuristic algorithms may operate either on a single solution candidate or on a set of solution candidates, the so called *population*. We briefly overview two main types of metaheuristic algorithms:

- **Trajectory-based:** Trajectory algorithms are local search techniques, that operate upon a single candidate solution, attempting to improve its quality until an optimal or near-optimal solution has been found [143]. A well known example of a local search techniques is *hill-climbing*. Hill climbing [143, 51] is initialised by selecting a solution candidate either at random or by exploiting some pre-existing domain knowledge. Then, the algorithm searches a neighbourhood of solutions for a fitter candidate, which becomes the current problem solution.

The process repeats until a satisfactory solution has been found, or until a predefined amount of time has elapsed. Hill climbing is prone to traps to local optima as the restriction of selecting only improved solution candidates limits the exploration ability of the search. An alternative is offered by *simulated annealing* which allows less fit individuals to be selected based on a gradually reducing probability [143]. In other words worsening moves are accepted but as the time passes it becomes less likely that a worsening move will be selected. The process enables the algorithm to avoid local optima.

- **Population-based:** As their name suggests, population-based algorithms optimise a population of solutions. The most widely used population-based algorithm is the *genetic algorithms (GAs)* [90, 80]. GAs are inspired by the biological process of evolution. The guiding principle is to create new offspring based on a current population of candidates, and then select the fittest candidates to survive and mate. A generic genetic algorithm routine is showed in Snippet 1.

Algorithm 1 A Generic Genetic Algorithm

1: **procedure**
2: *Set generation number,* $m := 0$
3: *Define initial population of candidate solutions,* $P(0)$
4: *Evaluate the fitness of each individual* $F_i(P(0))$ *in* $P(0)$
5: **loop**:
6: *Recombine* $P(m)$, $P(m) := R(P(m))$
7: *Mutate* $P(m)$, $P(m) := M(P(m))$
8: *Evaluate* $F(m)$
9: *Select* $P(m)$, $P(m) := M(S(m))$
10: $m := m + 1$
11: **exit**: *when stop condition satisfied*
12: **end loop**

A typical GA is initialised by a randomly chosen population. The iterations, so called *generations*, evaluate the fitness of the population applying the fitness function and then breed the fittest individuals to produce a new population [143]. The process

terminates when a population satisfies some pre-determined condition or a certain number of generations have been exceeded [90]. In many GA algorithms the fittest solutions in a population, so called the *elite*, are maintained unaltered in the new population. This encourages the elites to become parents and therefore attempt to improve the fitness of the children, though this can cause the population to prematurely converge on local optima.

As there usually exists no single global optimum, the output of GAs is a *Pareto optimal* set of solutions (also referred to as Pareto optimum) in the decision space [143]. This Pareto optimal set comprises individuals for which the improvement of one objective (e.g., low energy consumption costs) would inevitably lead to a deterioration of another objective (e.g., higher response times). Each Pareto solution is so called non-dominated, i.e., incomparable to the rest candidates of the Pareto set because no other solution has better values for all objectives. The image of the Pareto set to the decision space is referred to as Pareto front [215].

Genetic Operators

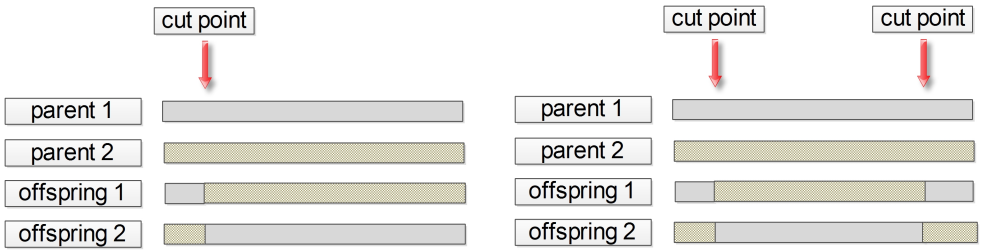


Figure 2.3: Illustration of single-point crossover (left) and two-point crossover (right).

Genetic operators are used to breed new solution candidates. The goal of genetic operators is to help discover new and possibly superior individuals. The breeding phase of GAs typically consists of the application of two genetic operators; *crossover* and *mutation*. The crossover operator ensures that features of the parents will be passed to the offspring promoting convergence to good solutions found so far (*ex-*

ploitation). The application of crossover produces two children from two parent individuals by mixing their genotypes. The most common crossover operators are *single-point* crossover and *two-point* crossover, shown in Figure 5.3. The mutation operator is then applied to the set of children produced through crossover. It makes stochastic changes to the offspring, to maintain diversity in the population, and avoid premature converge on a single area of the solution space (*exploration*). The probabilities of crossover and mutation are configurable and referred to as *crossover rate* and *mutation rate*. Depending of the particular representation that the search-algorithm is being applied to, different mutation and crossover operators are defined.

Selection

The goal of selection is to pick the best individuals of a generation. There are numerous techniques to select individuals from the population. In *fitness-proportionate* selection, an individual is selected for reproduction based on its fitness in comparison with the rest of the population [143]. However, fitness-proportionate selection has been criticised because fit individuals that appear early in the progression of the search lead the search to prematurely converge on local optima. [91]. To remedy this problem, several stochastic selection techniques have been proposed. In *roulette wheel* selection, individuals with higher fitness have higher probability of being selected [82]. A popular selection strategy is the *tournament selection* [143]. In tournament selection, an individual is randomly chosen from the population and is pairwise compared with a randomly selected opponent. The individual with highest fitness is the "winner". A fixed number n of tournaments take place, and the winner is being kept after each round. After n tournaments the new population has been selected, comprising the tournament "winners".

Different GAs employ different fitness assignment schemes to enable the selection of near-optimal solution candidates. The *non-domination rank* measure is used by

NSGA-II [57] to differentiate better from worse candidates. Each individual of the current Pareto set are assigned the best (lowest) rank. Then, all candidates of the Pareto set (i.e., that have been so far assigned the lowest rank) are removed and the Pareto set of the remaining candidates is assigned a rank of two and then are removed. This approach is repeated until all candidates are assigned a rank. Between two solutions with differing non-domination ranks, the solution with the lower rank will be selected. The *pareto strength*, used by SPEA [214], assigns to each individual in the population a real value $\in [0, 1)$, proportional to the population members it dominates. Then, the fitness of each an individual corresponds to the sum of the strength values of all candidates dominating the individual. Smaller fitness values are preferred during the selection process.

To discriminate between candidates with the same fitness additional measures are introduced to enable the selection of the promising individuals. The *crowding distance*, used by NSGA-II, measures the perimeter of the cuboid formed using the nearest neighbours of c as vertices. Another density measure introduced in SPEA2 [216], that calculates the density of a solution as the inverse of the distance to its k-th nearest neighbour.

The selection process in combination with the crossover genetic operator, can be seen as inducing *innovation* as good solutions are combined into potentially better solutions [199, 81]. On the other hand, the selection process in combination with the mutation operator, can be seen as evoking continual improvement to the population's fitness by introducing variation.

2.1.3 Statistical Regression Foundations

In this Section we detail the key concepts of statistical regression, that constitutes the necessary background for our surrogate models in Chapter 6. Overall, every regression analysis procedure involves the following steps [41]: *(i) Problem formulation:* the

question to be addressed must first be accurately defined, *(ii) Variables specification:* a set of variables relevant to explaining the response is identified, *(iii) Data collection:* data from the environment are observed, usually consisting of observations on n subjects. Each subject corresponds to a specified variable, *(iv) Regression model selection:* an initial model form (i.e., linear or non-linear) that could relate the response to predictors is specified, *(v) Model fitting:* The regression model parameters are estimated based on the collected data and *(vi) Validation:* Regression analysis is an interpretive process in which outputs are used diagnose, validate, criticize and modify the inputs as Figure 2.4 shows.

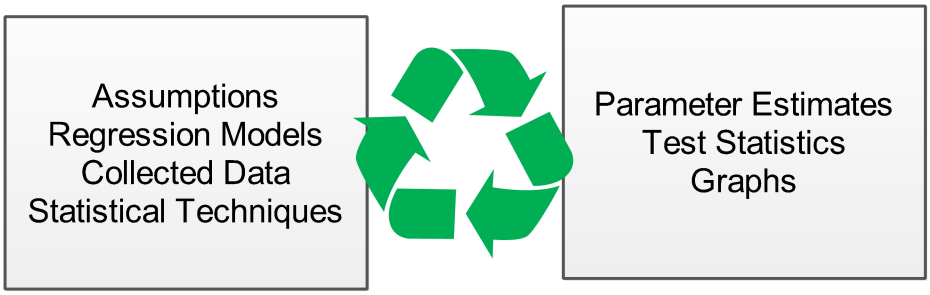


Figure 2.4: A schematic illustration of the iterative nature of regression processes. Assumptions are initially made about the models and tested using a series of test statistics and graphs. Based on this testing feedback, assumptions can be refined or changed continously till the desired results are achieved.

2.1.4 Regression Analysis

Regression analysis is a method for investigating functional relationships among variables [41]. The relationship is expressed in the form of an equation which models the effect of one or more *independent or predictor* variables $X_1, .., X_p$ on a *dependent or response* variable Y e.g., the effect of CPU request sizes on the request response time. When dealing with one response variable, the regression analysis is called *univariate* and in cases of two or more response variables the regression is called *multivariate*. Additionally, when we consider a single predictor the regression is called *simple*, otherwise *multiple*.

The true relationship between Y and $X_1, .., X_p$ can be approximated as [41]:

$$Y = f(X_1, X_2, ..., X_p) + \epsilon \tag{2.1}$$

where p is the number of predictors and ϵ represents a random error, accounting for the failure of the model to fit the data exactly. The model f can have linear or nonlinear form as described in the following.

The bias versus variance trade-off discussed in Chapter 5 in the context of multi-objective optimisation, appears in regression analysis as well. Balancing bias and variance is crucial towards selecting a regression model with good generalisation performance, i.e. good prediction capability on independent data.

Let us consider a response variable Y, a predictor X and a regression model $f(X)$ that has been estimated from a training dataset T. We assume $Y = f(X) + \epsilon$, where $E(\epsilon) = 0$ and $Var(\epsilon) = \sigma_\epsilon^2$. The loss function for measuring errors between Y and predicted outputs $f(X)$ is denoted by $L(Y, f(X))$. Typical choices are:

$$L(Y, f(X)) = \begin{cases} (Y - f(X))^2 & \text{squared error} \\ |Y - f(X)| & \text{absolute error} \end{cases}$$

The prediction error over the test sample T is called *test* or *generalisation error*:

$$ERR_T = E[L(Y, f(X))|T] \tag{2.2}$$

The generalisation error of a regression fit $f(\hat{X})$ at a data point $X = x_0$ using squared error loss can be decomposed as follows [92]

$$\begin{aligned} ERR(x_0) &= E[(Y - f(\hat{x}_0))^2 | X = x_0] \\ &= \sigma_\epsilon^2 + [Ef(\hat{x}_0) - f(\hat{x}_0)]^2 + E[f(\hat{x}_0) - Ef(x_0)]^2 \\ &= \sigma_\epsilon^2 + Bias^2(f(x_0)) + Var(f(x_0)) \\ &= \text{Irreducible Error} + Bias^2 + Variance \end{aligned} \tag{2.3}$$

From Equation 2.3 we observe that *bias* measures how far the model predictions are from the correct values, while *variance* indicates how much the predictions for a given point vary between different regression model realisations.

Training error ($e\bar{r}r$) is the average loss over the training sample:

$$e\bar{r}r = \frac{1}{N} \sum_{1}^{N} L(Y, f(X))$$

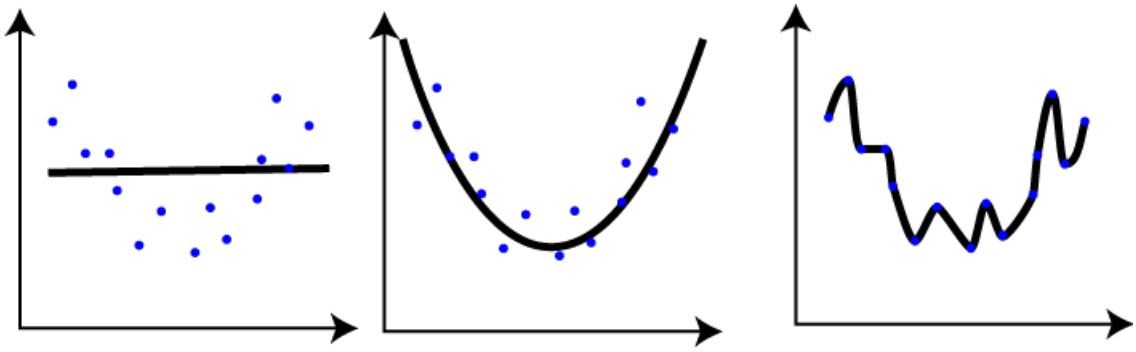


Figure 2.5: From Left to right: high bias, balanced bias-variance trade-off, high variance.

As a regression model becomes more and more complex it uses the training data more and is able to adapt to more complicated underlying structures. This means that there is a decrease in the bias and increase in the variance[71]. The training error consistently decreases with model complexity, typically dropping to zero if we increase the model complexity enough. However, a model with zero training error

is *overfit* to the training data and will typically generalize poorly. In contrast, if the model is not complex enough, it will *underfit* and may have large bias resulting in poor generalization. To visually represent the bias-variance trade-off, Figure 2.5 show examples of underfitted and overfitted models. In the left part of the Figure we observe a model that is too simple but the solutions are biased and do not fit the data. In the right part we observe the opposite problem; the model is very complex and becomes too sensitive to small changes in the data.

In the following we will present a series of widely used regression models. Our discussion starts from simple models such as linear regression and proceeds to more complex models such as Gaussian processes.

Linear Regression

In this regression technique, the relationship between a response Y and predictors $X_1, .., X_p$ is described by a linear model:

$$Y = \beta_0 + \beta_1 X_1 + ... + \beta_p X_p + \epsilon \tag{2.4}$$

where $\beta_1, ..., \beta_p$ are called *slopes*, β_0 *intercept* and ϵ is a random error. Together, $\beta_0, \beta_1, ..., \beta_p$ are called *regression coefficients*. The model building process derives the regression coefficients based on the available set of collected data observations.

The *residual* for observation i is defined as the difference between the observed (y_i) and predicted ($\hat{y}$) values. For a linear regression model the residual for observation i is given by

$$y_i - \hat{y}_i = y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_j \tag{2.5}$$

The popular *least squares method* estimates the coefficients, so as to minimise the sum of squares of the vertical distances from each data point to the regression line,

called *Residual Sum of Squares (RSS)* [92]:

$$RSS(\beta) = \sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_j)^2 \tag{2.6}$$

To find the minimum, the derivative of RSS is set to zero and solved for each coefficient. The residual sum of squares is given by [92]:

$$RSS = (y - X\beta)^T(y - X\beta) \tag{2.7}$$

where X denotes the $(N \times (p+1))$ vector of inputs and y the N vector of outputs in the training dataset

Differentiating with respect to β yields [92]:

$$\frac{\partial RSS}{\partial \beta} = -2X^T(y - X\beta) \tag{2.8}$$

Under the assumption that X has full column rank and hence X^TX is positive definite, the unique solution $\hat{\beta}$ is obtained by [92]

$$\begin{aligned} X^T(y - X\beta) &= 0 \\ \hat{\beta} &= (X^TX)^{-1}X^Ty \end{aligned} \tag{2.9}$$

If the columns of X are linearly dependent, X is not full rank. Then X^TX is singular and the least squares coefficients $\hat{\beta}$ are not uniquely defined.

Figure 2.6 shows the geometry of vertical distances from the observed data points to the line for a model with 2 predictors.

Assumptions

Linear models are based on the following assumptions [92]. Minor violations of the underlying assumptions do not compromise the conclusions drawn for the regression

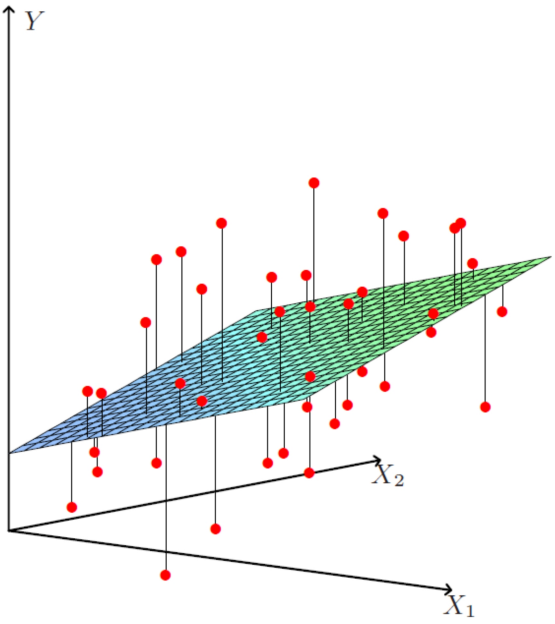


Figure 2.6: Linear least squares fitting [92]. We seek the linear function of X that minimizes the sum of squared residuals from Y.

analysis. However, gross violations of the model assumptions can seriously distort the conclusions.

- **Linearity:** The model that relates the response Y to predictors $X_1, X_2, ..., X_p$ is assumed to be linear in the regression parameters $\beta_0, \beta_1, ..., \beta_p$, conforming to the form [41]:

$$Y = \beta_0 + \beta_1 X_1 + ... + \beta_p X_p + \epsilon \tag{2.10}$$

  This implies that observation i will take the form

$$y_i = \beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip} + \epsilon_i, i = 1, 2, ..., n \tag{2.11}$$

- **Errors:** The errors $\epsilon_1, \epsilon_2, ..., \epsilon_n$ in 2.11 are assumed to be independently and and identically distributed random variables with zero mean and common variance σ^2. The fact that all the errors $\epsilon_1, \epsilon_2, ..., \epsilon_n$ have the same variance is also known as the *homoscedasticity* assumption. The fact that all errors are independent

of each other means that their pairwise covariances A.1 will equal zero, and is known as the *independent-errors* assumption [41].

- **Predictors:** The predictor variables $X_1, X_2, ..., X_p$ are assumed to be non random. The observations $x_{1j}, x_{2j}, ..., x_{nj}, j \in [1, p]$ of the predictors are fixed and measured without error. The latter assumption is hardly ever satisfied in practise. The magnitude of the violation effects mainly depends on the standard deviation of the errors of the measurements and the correlation among the errors. The predictor variables are also assumed to be linearly independent of each other. If this assumption does not hold, the problem is referred to as the *collinearity problem* [41].

- **Observations:** All observations are considered equally reliable and have approximately equal role in determining the regression results and in influencing conclusions [41].

The main virtues of the linear model are its simplicity of implementation and interpretability. Its main drawback is that allows limited flexibility; if the relationship between input and output cannot reasonably be approximated by a linear function, the model will generalise poorly.

Classification and Regression Trees (CART)

Tree-based methods partition the feature space into a set of rectangles, and then fit a simple model (e.g., a constant) in each one. Considering p predictors and a response Y, the feature space is initially split in two regions, and the response is modelled by the mean of Y in each region. Then one or both of these regions are split into two more regions, and this process is continued, until a predefined stopping rule is applied. Figure 2.7 illustrates a tree model with 5 regions: initially X_1 is split at $X_1 = t_1$,

then region $X_1 \leq t_1$ is split at t_2 and the region $X_1 > t_1$ is split at t_3. Finally the region $X_1 > t_3$ is split at t_4.

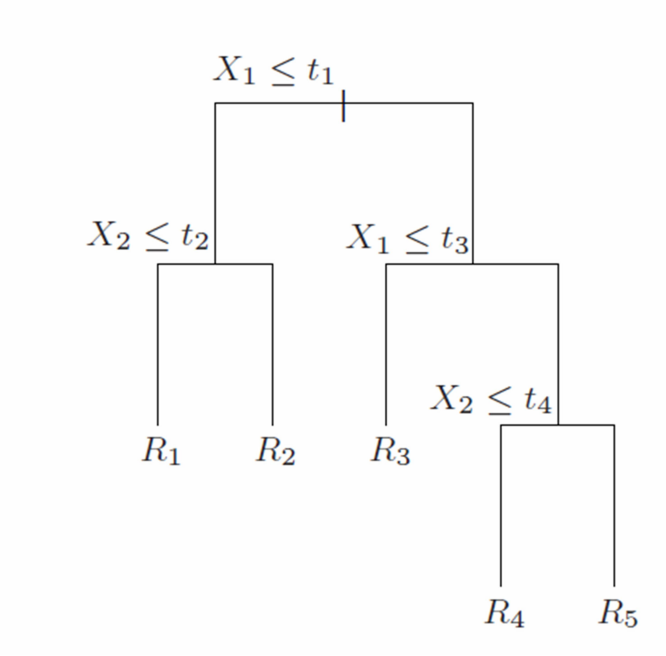


Figure 2.7: Classification and Regression Trees (CART) with 5 partitions [92].

Assuming M partitioned regions and a constant c_m to model the response at each partition, a Classification and Regression Tree (CART) is formalised as **??**:

$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m) \tag{2.12}$$

where I denotes each region.

Adopting as a partition criterion the minimisation of the sum of squares $\sum(y_i - f(x_i))$, the best approximation for $\hat{c_m}$ is the average of y_i in region R_m. However, finding the best binary partitions in terms of minimum sum of squares is computationally infeasible. Hence CART forms partitions based on a greedy algorithm. The algorithm seeks a splitting variable j and a splitting point s that define a pair of half planes:

$$R_1(j,s) = \{X|X \leq s\} \text{ and } R_2(j,s) = \{X|X > s\} \tag{2.13}$$

such that:

$$\min_{j,s} \; [\; \min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2] \tag{2.14}$$

For each j, s the solution for the inner minimisation equation is:

$$\hat{c_1} = ave(y_i|x_i \in R_1(j,s)) \text{ and } \hat{c_2} = ave(y_i|x_i \in R_2(j,s)) \tag{2.15}$$

The size of the tree is adaptively decided based on the available data. The most popular strategy typically grows the tree till a predefined minimum node size is reached. To avoid overfitting the initial large tree T_0 is reduced using *cost-complexity pruning*, which results to a subtree $T \subset T_0$. T is obtained by pruning T_0 i.e., collapsing any number of its non terminal nodes. The idea is find the subtree T_0 that minimises 2.16. Let us denote $|T|$ the number of terminal nodes in T and index terminal nodes by m, where node m represents region R_m. Letting:

$$N_m = \#\{x_i \in R_m\}$$

$$\hat{c_m} = \frac{1}{N_m} \sum_{x_i \in R_m} y_i$$

$$Q_m(T) = \frac{1}{N_m} \sum_{x_i \in R_m} (y_i - c_m)^2$$

The cost complexity criterion is defined as:

$$C_\alpha(T) = \sum_{m=1}^{T} N_m Q_m(T) + \alpha T \tag{2.16}$$

The parameter α in 2.16 governs the trade-off between the tree size and its goodness of fit to the data. Large values of α result in smaller trees and conversely for smaller values of α. The pruning process successively collapses the internal nodes resulting in the smallest increase in $\sum_m N_m Q_m(T)$.

Multivaritate Adaptive Regression Splines

Multivaritate Adaptive Regression Splines (MARS) [92] can be viewed as a generalisation of linear regression or a modification of the CART method to improve the latter's performance. MARS models consist of piecewise linear functions in the form $(x-t)_+$ and $(t-x)_+$, defined as:

$$(x-t)_+ = \begin{cases} x-t, & \text{if } x > t \\ 0, & \text{otherwise} \end{cases} \qquad (t-x)_+ = \begin{cases} t-x, & \text{if } x < t \\ 0, & \text{otherwise} \end{cases}$$

where the $+$ denotes the positive part and the value t is called *the knot*. Figure 2.8 shows an example of piecewise linear functions with a knot $t = 0.5$.

The idea is to form piecewise functions for each predictor $X_j, j \in [1, p]$ with knots at each observation $xji, i \in [1, n]$. Therefore, the collection of basis functions is

$$C = (X_j - t)_+, (t - X_j)_+$$

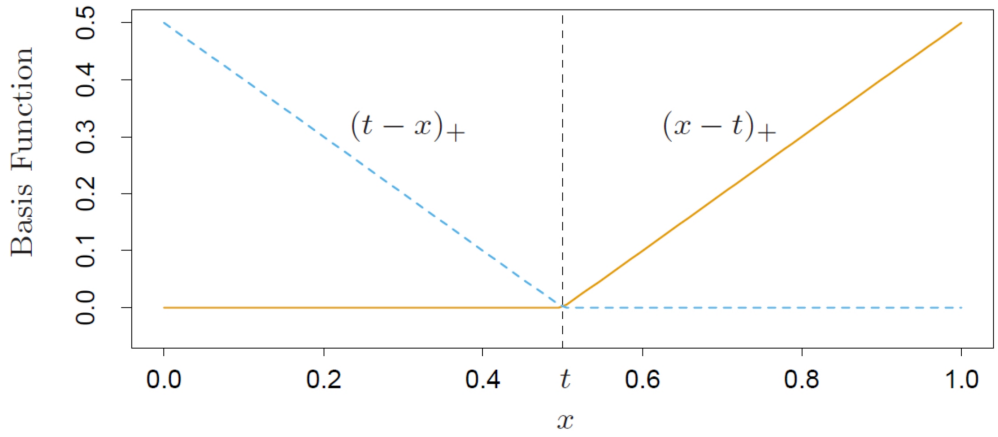


Figure 2.8: Piecewise linear functions with knot $t = 0.5$ [92].

where $t \in x_{1j}, x_{2j}, ..., x_{Nj}$ and $j \in [1, p]$. At the end of the fitting process the MARS model is a weighted sum of hinge functions and constant coefficients β_i

$$f(X) = \beta_0 + \sum_{i=1}^{M} \beta_i h_i(X) \tag{2.17}$$

where $h_i(X)$ is a function in C, or a product of two or more such functions. The coefficient parameters β_i are estimated by the method of least squares as for the linear regression model. The model is constructed iteratively, starting with a null model with only the intercept β_0. The algorithm iteratively adds a hinge function to the model, that produces the largest decrease in training error. The process is continued until the model set M contains a predefined maximum number of terms. This model typically overfits the data, so a backwards deletion process is finally applied. The term whose removal causes the smallest increase in residual squared error is deleted from the model at each stage.

Support Vector Regression

Let us assume a training dataset x comprised of N pairs $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$ with $x_1 \in (\mathbb{X} = \mathbb{R}^p)$. The modelling aim in Support Vector Regression (SVR) is to

identify a function $f(x)$ that has at most ϵ deviation from the actually obtained targets y_i for all the training data and at the same time as flat as possible [178, 92].

The case of a linear f takes the form:

$$f(x) = \langle \omega, x \rangle + b \ , \ \omega \in \mathbb{X}, b \in \mathbb{R} \tag{2.18}$$

where $\langle \dots \rangle$ denotes the dot product in $\mathbb{X}$. Flatness in 2.18 means small ω. One way to achieve flatness is to minimise the the Euclidean norm i.e., $||\omega||^2$. The problem is formulated as [92]:

$$\begin{aligned} &\text{minimise } \frac{1}{2}||\omega||^2 \\ &\text{subject to } \begin{cases} y_i - \langle \omega, x \rangle - b \leq \epsilon \\ \langle \omega, x \rangle + b - y_i \leq \epsilon \end{cases} \end{aligned} \tag{2.19}$$

The above optimisation problem is feasible on cases when a function f exists that approximates all pairs (x_i, y_i). Sometimes this is not the case and errors are allowed introducing slack variables ξ, ξ^* to relax the constraints of the optimisation problem:

$$\begin{aligned} &\text{minimise } \frac{1}{2}||\omega||^2 + C \sum_{i=1}^{l} (\xi + \xi^*) \\ &\text{subject to } \begin{cases} y_i - \langle \omega, x \rangle - b \leq \epsilon + \xi \\ \langle \omega, x \rangle + b - y_i \leq \epsilon + \xi^* \\ \xi, \xi^* > 0 \end{cases} \end{aligned} \tag{2.20}$$

where C is a constant that determines the trade-off between flatness of f and the amount tolerated deviations from ϵ. The geometry of slack variables is shown in Figure 2.9. The equations are finally solved for ω and b utilizing Lagrange multipliers and exploiting the Karush-Kuhn-Tucker conditions [92, 178].

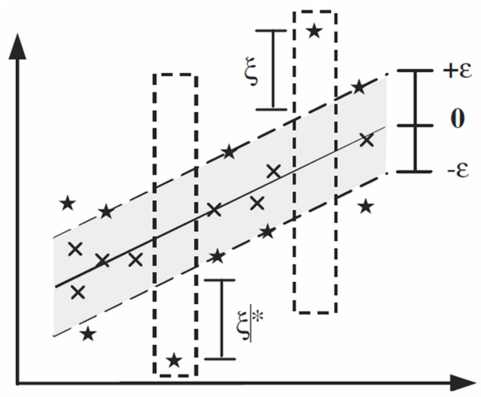


Figure 2.9: Slack variables [45].

SVR can be made nonlinear by preprocessing the training data x_i applying a map $\Phi : \mathbb{X} \longrightarrow \mathbb{F}$ into a higher dimensional feature space $\mathbb{F}$. The same SVR algorithm is then applied in $\mathbb{F}$. SVR only depends on dot products between the training data x_i, hence it suffices to know $k(x, x') = \langle\ \Phi(x), \Phi(x')\rangle$ rather than Φ explicitly. The function $k(x, x')$ is so called *kernel* function. Some common kernel choices are presented in Table 2.1.

Kernel	**Formula**
d-th degree polynomial	$k(x, x') = (1 + \langle x, x'\rangle)^d$
Radial basis	$k(x, x') = exp(-\gamma\|\|x - x'\|\|^2)$
Neural network	$k(x, x') = tanh(\kappa_1 \langle x, x'\rangle) + \kappa_2)$

Table 2.1: Common kernel choices.

Gaussian Processes

Rather than making assumptions about the characteristics of the underlying modelling functions, Gaussian Processes (GPs) for regression assign a *prior* probability to every possible function [92]. These probabilities represent prior beliefs over the kinds of functions expected to be observed before seeing the data. Assuming that a dataset D becomes available, higher preference is given to the functions close the data points.

As Figure 2.10 illustrates, the combination of prior and data leads to the *posterior* distribution over functions [165].

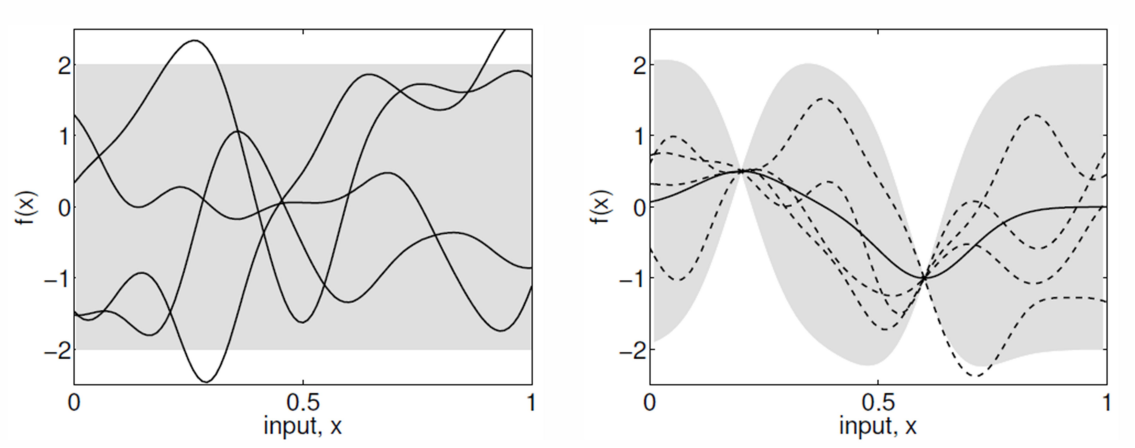


Figure 2.10: Left: samples of prior distributions without observed data points. Right: samples of posterior distribution after 2 data points have been observed. [165].

A Gaussian Process (GP) is a collection of random variables, any finite number of which have a joint Gaussian distribution. A GP is completely specified by its mean $m(x)$ and covariance function $K(x, x')$:

$$f(x) \sim GP(m(x), K(x, x')) \tag{2.21}$$

For simplicity a mean function of zero is usually assumed. Assuming n noise free observations $\{(x_i, f_i)|i = 1, \ldots, n\}$, the joint distribution of training outputs f and test outputs f_*:

$$\begin{bmatrix} f \\ f_* \end{bmatrix} \sim \mathcal{N}\left(0 \quad \begin{bmatrix} K(X, X) & K(X, X_*) \\ K(X_*, X) & K(X_*, X_*) \end{bmatrix}\right) \tag{2.22}$$

where for n_* new test points $K(X, X_*)$ denotes the $n \times n_*$ matrix of covariances evaluated for all pairs of training and test points, and similarly for the entries $K(X, X)$, $K(X_*, X_*)$ and $K(X_*, X)$. The predicted outputs f_* are given by the conditional probability:

$$
\begin{aligned}
f_*|X_*, X, f \ &\sim \ \mathcal{N}(K(X_*, X)K(X, X)^{-1}f, K(X_*, X_*) \\
&- \ K(X_*, X)K(X, X)^{-1}K(X, X_*))
\end{aligned}
\tag{2.23}
$$

Model Selection and Assessment

This section discusses common strategies for the evaluation of the generalisation performance of regression models. This performance assessment is very important since it drives the choice of learning method or model and gives a measure of the quality of the ultimately chosen model. The process of performance assessment involves two separate steps: (i) *Model selection* which involves the performance assessment of a series of different models on test data in order to decide the best and (ii) *Model assessment* where the prediction error of the best model is estimated on new data [92].

In data-rich situations, the most common performance assessment strategy randomly divides the dataset into three parts; a training set, a validation set and a test [92] set as illustrated in Figure 2.11.

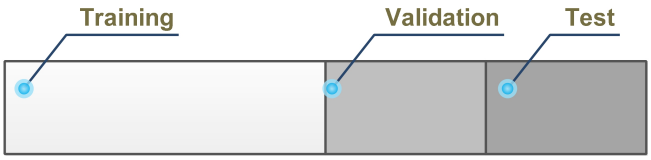


Figure 2.11: Data split example.

The training set is used to fit the models while the validation set is used to estimate prediction error for model selection. Finally the test set is used for the assessment of the generalisation error of the final chosen model. No general rule applies on how to choose the number of observations in each of the three parts. A typical split shown in Figure might be 50% for training, and 25% each for validation and testing [92].

K fold cross-validation is probably the most widely used assessment strategy, particularly for situations where data is scarce and practitioners cannot afford to set aside a test set [92]. In K-fold cross-validation, the initial dataset is split into K folds. Typical choices for K are 5 and 10. For the k-th part, we fit the model to the other $k-1$ parts of the data, and calculate the prediction error of the fitted model when predicting the kth part of the data. The method iterates through $k = 1, 2, \ldots, K$ and finally combines the K estimates of the prediction error. Figure 2.12 illustrates the 10-fold cross-validation process.

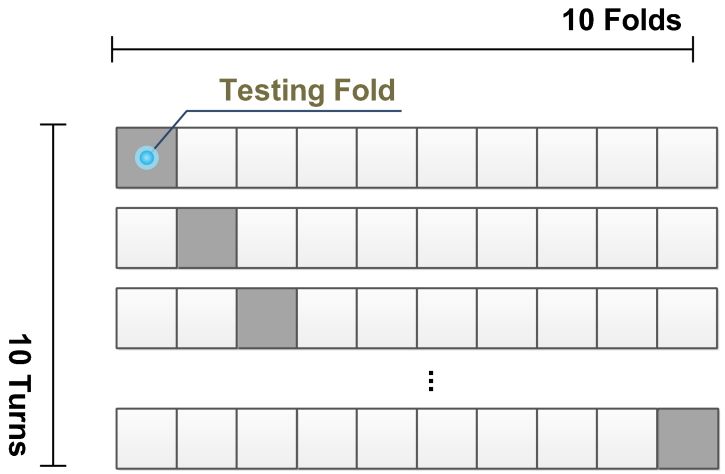


Figure 2.12: 10-fold cross-validation.

Cross-validation ensures that every example of the original set has the same chance of appearing in the training as well as test sets.

Performance Evaluation Metrics

Numerous metrics may be used to measure the prediction error between the forecasts and corresponding observations [208, 107, 49, 197]. Assuming a dataset of n observations, we present in the following the most common metrics:

- *Coefficient of Determination (R^2):* It gives information about the *goodness of fit* of the model, indicating the percentage of variability in the response variable

Y that can be explained by the selected regression model:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}{\sum_{i=1}^{n}(Y_i - \frac{1}{n}\sum Y_i)^2} \tag{2.24}$$

R^2 ranges from zero to one, with one indicating a perfect fit. Improvement in the regression model results in proportional increases in R^2. A pitfall though, is that it increases as more predictors are added to the regression model, which may lead to false indication of improved performance. This increase in R^2 is artificial when useless predictors are added to the model, i.e., predictors that do not improve the model's fit but only add unnecessary complexity. To remedy this R^2 must be studied in combination with other metrics discussed in the following.

- *Mean Absolute Error (MAE):* MAE calculates the absolute errors between the observed and predicted values as follows:

$$MAE = \sum_{i=1}^{n} \frac{\left|Y_i - \hat{Y}_i\right|}{n} \tag{2.25}$$

MAE has the same unit as the original data, and it can only be compared between models whose errors are measured in the same units. A lower value of MAE implies a better fit of the prediction model.

- *Mean Squared Error (MSE):* MSE measures the average of the squares of the prediction errors and incorporates both the bias and variance of a model as discussed in Section 2.1.4. A lower value of MSE indicates a better fit.

$$MSE = \frac{\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}{n} \tag{2.26}$$

- *Root Mean Squared Error (RMSE):* RMSE is the standard deviation of errors, measuring the typical spread of data around the regression line.

$$RMSE = \sqrt{\frac{\sum i = 1^n (Y_i - \hat{Y}_i)^2}{n}} \tag{2.27}$$

  RMSE is measured in the same units as the data. Lower values indicate better predictive performance. RMSE is the most widely used measure of prediction error and a good measure of how accurately the model predicts the response. In contrast with the R^2 metric which is a relative measure of fit, RMSE is an absolute measure of fit. This means that there is no accepted cut-off value with which to decide whether an estimate is acceptable or not, but the metric is useful when comparing predictive models for the same problem.

- *Mean Absolute Percent Error (MAPE):* MAPE offers a weighted error measure, expressing a generic percentage term in comparison to the previous measures that are expressed in same units as the observed data:

$$MAPE = \frac{1}{N} \sum^{N} \left| \frac{Y_i - \hat{Y}_i}{Y_i} \right| \tag{2.28}$$

  MAPE measures how higher or lower are the predictions in comparison to the actual data, e.g., 20% MAPE means that the predictions will be within a range of 20% higher or lower than actual.

An efficient regression model complexity should trade bias off with variance in a way that optimises the aforementioned performance metrics. As more and more parameters are added to the model the complexity rises and high variance becomes a concern while bias steadily falls. At the same time the training error tends to decrease but the model does not generalise well on new data (i.e., large test error occurs). In contrast a simplistic model has increased bias, resulting again in poor generalisation.

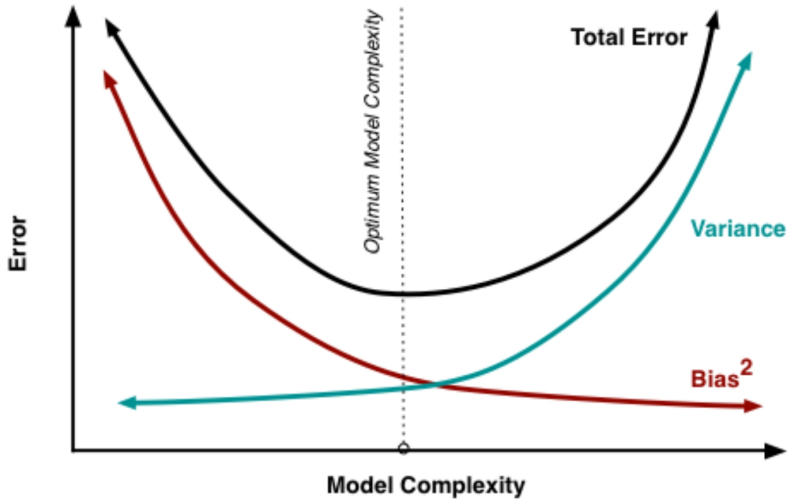


Figure 2.13: Bias and variance as a function of model complexity [71].

As Figure 2.13 suggests golden section of any model is the level of complexity at which the increase in the bias value is equivalent to the reduction in the variance value[71].

2.2 Reactive Methods

In this section we present an overview of reactive resources provisioning methodologies in virtualised and cloud environments, that mainly focus on achieving optimised configurations without having a priori knowledge about future environment changes such as workload fluctuations. An optimal configuration is considered to best balance different non-functional goals. Common optimisation goals in the current literature include minimisation of energy/power consumption and maximisation of services performance, that are in mutual conflict with each other; e.g., while additional CPU increases the processing capacity of the system and therefore more requests can be served it also increases the capital and operating costs of the datacenter and vice versa [96, 99, 192].

A typical method to achieve minimised energy/power costs is the *consolidation* of hardware [117]. Before the advent of virtualisation technology and the first hypervisor in 2003 [16], a hardware server was typically dedicated to each single application [121]. Therefore business growth resulted to the phenomenon of server and storage "sprawl", i.e., the proliferation of underutilised hardware. Virtualised and cloud datacenters consume large amounts of electrical energy contributing to high operating costs and carbon footprints to the environment [21]. Consolidation is used to reduce the number of active PMs in datacenters by grouping together multiple applications executing in VMs, in the minimal amount of hardware nodes. Consolidation is facilitated via VM migrations. Not only does it facilitate reduced operating costs, but also is used as a mean towards green computing [151, 138]. However lowering the energy usage of datacenters is a complex issue because computing applications and data are growing so quickly that increasingly larger servers and storage are needed to process them fast enough [35]. Consolidation is therefore essential for clouds to guarantee that future application growth will be sustainable. Otherwise, cloud computing with increasingly pervasive front end client devises interacting with back end servers will cause an escalation of energy usage [21]. To address these issues cloud resources such as VMs and PMs must be allocated not only to satisfy the applications Quality of Service (QoS) but also to reduce energy usage. The problem of identifying a minimal configuration that balances the trade-offs of maximum QoS and minimum energy use is the biggest challenge of consolidation efforts, and is identified as NP-hard [96, 201].

Khanna et al. [121] present a framework for consolidation at runtime. The authors assume a homogeneous datacenter, where all nodes have equal CPU and memory capabilities. First, resource utilisation rates are mapped to (Service Level Agreement) SLAs compliance ratios. Therefore, exceeding predefined utilisation levels is associated with SLA violations. A performance violation triggers the proposed resources re-allocation algorithm, that updates the current datacenter configuration by iden-

tifying the most computationally efficient migrations. This is based on the authors' observation that migrating VMs from low utilised PMs is cheaper than migrating from highly utilised PMs. The algorithm sorts the PMs in a decreasing utilisation order. In the case of SLA violations, a VM from the lowest utilised PM is migrated to the highest utilised PM, with enough capacity to host it. If the action does not solve the issue, a new PM is activated. The algorithm is evaluated on a sample environment of 11 VMs and 3 PMs while the convergence time is not reported.

This approach is restricted to a single degree of freedom i.e VM migrations. The single adaptation point limits the design space exploration achieved during optimisation. Hence the proposed framework cannot allow fine-grained tuning of non functional requirements.

Entropy [96] is a consolidation framework that operates in two phases; the first allocates the available VMs to the minimum amount of PMs and the second optimises the initial plan by employing a minimum amount of migrations. The authors posit the identification of globally optimum configuration solutions. To this end constraint programming is used, over heuristics to avoid traps to local optima. The proposed reconfiguration manager is triggered each time a VM changes state (i.e,. from active to inactive). The optimiser initially enumerates every possible configuration. The configuration with minimum servers that satisfies SLAs (i.e., is viable) is considered globally optimal. Towards improving the search time, non viable configurations are rejected using time-outs, upper and lower bounds on the number of total VMs that can be hosted from a PM and VMs are classified together according to their similarity (i.e., same capabilities). The second phase of the optimisation process improves the initial configuration based on a refined set of constraints, that achieve a minimal number of migration actions. Based on the authors' observation that the migration overhead depends on the migration duration, which is dependent on the migrating

VM memory size, a constraint satisfaction problem is formed restricting the memory consumption of the migrations in the overall reconfiguration process.

Entropy introduces valuable concepts and ideas for engineering consolidated datacenters, such as the association of the VM memory size with the total migration overhead. However the authors do not elaborate on the convergence time of the optimisation algorithm. Exhaustive exploration of the configurations design space is computationally expensive and therefore limits the adaptability of the system at runtime.

"Shirako" [83, 106] is a resources management framework aiming to provide automated and dynamic adaptations to changing workloads in virtualised environments. To achieve adaptations at real time, the authors employ bin-packing heuristics to flexibly reconfigure the system. For example the utilised "worst-fit" algorithm places a VM to the emptiest PM. The authors also focus on reconfiguring the system with the minimum possible amount of VM migrations to avoid services downtime. Therefore VM migrations are only instantiated if there is no free space to place new VMs after the execution of the bin-packing heuristics.

Similar to Khanna et al. [121] the authors introduce a single degree of freedom, that results in coarse-grained design space exploration. An advantage of the introduction of a single degree of freedom is the simplification of the optimisation process by reducing the possible reconfiguration paths. However the authors do not evaluate the quality of the achieved configuration solutions. Hence the gains of the problem abstraction are unclear.

Verma et al. propose "pMapper" [192], a framework that minimises power consumption in virtualised systems, subject to predefined performance SLAs. The authors aim to minimise the amount of migrations in the datacenter as well, since each migration requires additional CPU and memory to complete and therefore consumes additional power. The authors resort to a series of local search algorithms such as

the min Power Parity (mPP). The mPP algorithm first computes target utilization thresholds for each PM towards guaranteeing the predefined performance SLAs, and then executes a first-fit-decreasing greedy bin packing algorithm to tightly place the VMs to the PMs. The first-fit-decreasing algorithm sorts the existing VMs in decreasing order based on their size and places each VM to the first PM with sufficient capacity to accommodate it. Another algorithm, namely the incremental First Fit Decreasing (iFFD) takes effect next. The iFFD algorithm identifies the PMs, whose resource demands still exceeds the utilisation thresholds. To free space capacity in the heavily loaded PMs, VM migrations are triggered. Under the authors' assumption that the impact a of VM migration in performance is only dependent on the VM characteristics, the smallest sized VMs are migrated to minimise downtime effects.

A similar greedy packing methodology for energy efficient cloud configurations is "EnaCloud" [132]. A variation of first-fit-decreasing greedy bin packing algorithm is used. The difference is that each time a new workload arrives, the heuristic does not simply place the request to the first VM that can accommodate it. Instead the heuristic tries to displace already packed workloads that are smaller than the newcomer, with the newcomer. The smaller workloads are then reinserted in a similar manner. Based on the observation that smaller workloads are easier to be inserted in to gaps, the algorithm aims to tightly pack applications to the existing infrastructure without activating new nodes, and therefore regulating the total energy consumption of the system.

The aforementioned pMapper and EnaCloud frameworks both rely on the use of greedy bin packing heuristics. An advantage of greedy optimisation approaches is the simplification of the underlying problem. However the authors do not elaborate on the trade-off between the achieved quality of the optimised configuration solutions and the timeliness of the optimisation process. Hence it remains unknown how well similar frameworks can be used in real world scenarios.

Van et al. are concerned with the problem of SLAs satisfaction and resources management in cloud environments [187]. The problem is divided in two stages; the *provisioning* and the *placement* stage. The first stage is responsible for allocating resource capacity in the form of VMs to the applications. The authors resort to utility functions to measure the current level of applications' satisfaction based on current workloads and available resources. The utilities will drive the required resource needs per application. The placement stage is responsible for allocating to the VMs the required CPU and RAM resources of the PMs, constrained by the total capacity of the cloud. Each stage is described as a constraint satisfaction problem and solved by exhaustive search as in [96]. Due to the complexity of the exhaustive search, the approach is evaluated in a small cloud of 4 PMs.

Van et al. hypothesise that the cloud applications' behaviour is explained with the introduced utilities functions. However it remains unclear how the utility functions were derived and whether these can reflect runtime system variability attributes. Furthermore, the high computation time of the algorithm limits the ability of the system to react to changes in a timely fashion. Exhaustive optimisation approaches are not appropriate for cloud systems, where the inherent elasticity requirement makes runtime adaptation critical.

Yazir at al. propose a framework for dynamic resource allocation in cloud environments [207]. The authors' primary objective is avoid under-utilisation and over-utilisation. The first causes unnecessary costs due to idle active servers, that still consume up to 60% of their peak power consumption [146]. The second causes high application response times and therefore may result in SLA violations. To remedy the above, agents are coupled with every PM in the cloud to monitor its state. If an anomaly (i.e., over-utilisation or under-utilisation) is observed, corrective action is taken by initiating an optimal number of VM migrations using multi-criteria decision analysis. Three main criteria are used to evaluate each possible action; CPU, mem-

ory and bandwidth consumption. Predefined weights are assigned to prioritise each criterion. Next, all possible alternative actions are evaluated and pair-wise compared to identify the best sequence of migration actions.

A limitation of the approach is the introduction of predefined weights to prioritise the optimisation criteria. The weight values have been decided offline, at design time and therefore do not provide the means to adapt to unpredictable environment conditions.

Ferreto et al. [70] note that a plethora of consolidation methodologies rely on greedy VM migration, which might result in unpredictable applications performance degradations. The authors propose a Linear Programming (LP) formulation, that is re-solved periodically, and restricts VMs with steady resources usage from being migrated. According to the authors this policy assists hotspots resolution, without triggering unnecessary disruptions to properly functioning VMs.

A similar approach towards cost-efficient cloud configurations is proposed by Papagianni et al. [159]. As before, the authors' motivation is to optimally allocate user requests to virtualised resources. Each request is modelled as an undirected graph. Nodes in the graph indicate resources i.e., CPU, memory, storage while links are associated with bandwidth capacity. The resource allocation problem of mapping requests to the cloud VMs is formulated as a mixed integer programming problem. The objective is to minimise the amount of allocated resources in terms of nodes and links in the graph. To solve the problem a randomised rounding technique and shortest path algorithm are employed.

Breitgand et al. [53] also contribute with a framework for cost minimisation in cloud configurations. The authors aim to minimise revenue losses in the provider's side, due to SLA violation penalties and unnecessary VM migrations. The problem is formalised as an integer linear program. Additional constraints describe placement restrictions for VMs, towards minimising the number of allowed VM migrations. Bre-

itgand et al. differentiate their position from Chaisiri et al. above, noting that "an exact solution would be impractical since more than 10 million variables would have to be tuned". Therefore the authors apply a greedy packing heuristic to simplify the problem. The heuristic identifies a configuration plan, that achieves the highest PMs utilisation while respecting the predefined placement constraints.

The aforementioned contributions by Breitgand et al. [53], Papagianni et al. [159] and Ferreto et al. [70] use of integer linear programming for optimisation. The inherent assumption of linearity may hamper the applicability of the framework because it is seldom verified in real world cloud environments.

Jung et al. address the problem of online resources consolidation in cloud datacenters [143]. Online consolidation is critical, to achieve satisfactory performance in the cloud under changing workload conditions. The authors differentiate their approach from previous works, by explicitly considering the additional costs of adaptation actions such as VM migrations and switching PMs/VMs on and off. According to the authors' rationale, expensive adaptation actions (e.g., switching on new hosts) should not be preferred when the workload is rapidly changing, because the investment made will never be recouped as the system is continuously evolving. The proposed optimisation methodology is two-fold. First a bin-packing heuristic is used to find configurations that balance the applications' performance and power consumption trade-offs. The solution output is considered optimal. Next a search graph is constructed where edges are the adaptation actions and vertices are the possible configurations. The goal is to find the cheapest path leading to the optimal solution. The adaptation costs are calculated based on experimental offline estimations. As the search time increases exponentially with the size of the graph, the search space is reduced by grouping together configurations based on their similarity.

Jung et al. provide valuable insights on the importance of modeling the transient adaptation costs, that have been assumed as negligible by the majority of existing

literature. However contrary to the authors' assumption, the output of the initially applied greedy heuristic provides no guarantees that the achieved solution is near-optimal. Furthermore the adaptation costs represented in the proposed search graph, have been estimated empirically offline, at design time. Besides cost-inefficiency, if the design-time assumptions do not hold, static system configurations may become inefficient and the adaptation algorithm may stop behaving as expected. Overall we observed that the introduced assumption may hamper the generalisation ability of the proposed solution.

Ardagna and Gibilisco [9] formulate a mixed integer linear program approach to assist cloud users find a minimum cost cloud configuration to host a given application. Each VM is modelled a a M/G/1 queue and an extension of the palladio performance modelling language [20] is used to estimate the response time of each VM by analytically solving a layered queueing network. A VM configuration with minimum cost of usage, that honours pre-defined response time level constraints is selected. To further optimise the solution obtained, the authors apply stochastic local search to further fine tune the number and type of selected VMs.

A shortcoming of the proposed approach is that analytic models, like queuing networks, must be defined and tuned at design time and therefore hardly adapt to unpredictable configurations. Furthermore white-box modeling can be limiting in the cloud domain, because of the legal barriers between service and infrastructure providers [149].

Threshold-based policies are popular in current industrial applications as they are simple and intuitive to understand. They are applied by defining lower and uppoer limits on target metrics such as resource utilisation. Behaviours outside the ranges trigger reconfiguration actions. Threshold-based approaches are currently implemented by many companies such as Amazon [1] and Rightscale [2] [77]. The threshold

[1] `http://aws.amazon.com/autoscaling/`
[2] `http://www.rightscale.com/`

rules are typically empirically decided at design time, that limits the adaptability of reconfigurations to unpredictable runtime conditions. Furthermore as rules are typically empirical they can be coarse-grained and therefore imprecise.

Durteilh et al. [62] present a resource scaling approach based on static thresholds, that picture the desired system performance (i.e., expected average response time). A control-loop is executed at specific time intervals that monitors the current system state. In case the thresholds are violated corrective action is taken by means of allocating or deallocating a fixed amount of VMs. The uncertainty of environmental changes is a challenge for the approach, because the time intervals when the controller takes action do not always comply with the pace of environment changes, leading to instability.

Belaglazov et al. [21] propose a threshold-based energy-aware methodology for the management of cloud datacenter resources. CPU utilisation is considered the main driver for power consumption and it is typically proportional to the system load. Based on this observation, the authors' goal is to keep the datacenter CPU utilisation within a set of predefined upper and lower thresholds, that can guarantee both SLA compliance and energy savings. The proposed allocation scheme operates in two steps; the first assigned new resource requests to VMs and maps the VMs to PMs. The second ensures that the CPU utilisation stays withing the predefined thresholds. If the utilisation falls below the lower threshold, all VMs will have to be migrated to another PM, and the current host will be switched-off to save energy. If the utilisation exceeds the upper threshold, some VMs will be migrated to reduce the utilisation. This way free resources are preserved in the PMs, to avoid SLA violations due to consolidation in cases when the applications' resource demand increases. In any case the minimum amount of migrations is triggered. Belaglazov et al. further expanded their work by proposing adaptive utilisation thresholds for the cloud hosts [22], since static thresholds utilised in [21] are not suitable for the dynamic cloud environments. The

auto-adjustment of utilisation thresholds is based on statistical analysis of historical data collected during the lifetime of VMs. The main idea of the proposed adaptive-threshold algorithm is to adjust the value of the upper utilisation threshold depending on the strength of the deviation of the CPU utilisation. According to the authors, a high deviation indicates a high likelihood that the CPU utilisation will reach 100% and cause SLA violations. To avoid over-utilisation, the upper utilisation threshold is set to a lower value.

Rule-based approaches extend threshold-based solutions by considering different types of events and allowing rules to trigger actions following the common ECA (event, condition, action) paradigm. A rule-based approach for cloud resources reconfiguration is proposed by Merino et al. [167]. The authors leave full control for users to express their own rules and trigger appropriate reconfiguration actions according to customised performance preference. The possible reconfiguration rules include resizing VMs to increase ore decrease their computational capabilities, adding new VM instances or migrating services to another cloud provider. An example of the proposed ECA scheme is e.g., "if more than 50 tasks execute on a particular VM, replace the VM with a bigger one."

Evolutionary metaheuristics is a category of randomised search algorithms that are gaining increasing momentum in the cloud community. Metaheuristic search algorithms optimise either a single solution or a set of solutions, known as a population. Each candidate solution is evaluated to calculate its fitness a measure of how close that solution comes to solving the problem [199, 89]. Below we summarise a series of current works that position the use of evolutionary metaheurstics to solve the problem of optimal resources allocation in the cloud.

One of the first contributions was proposed by Zeng and Ye [205]. The authors aim to optimise the energy efficiency of cloud systems. A multi-constraint objective function is formulated that represents the goal of minimising the active servers in the

system while meeting hosted services performance requirements. The evolutionary algorithm NSGA-II is then deployed to search for near-optimal resource combinations.

Xu et al. [204] use the Grouping Genetic Algorithm (GGA) to simultaneously optimise the possibly conflicting objectives of resources usage, power consumption and thermal dissipation costs. GGA groups configuration solutions into a collection of mutually disjoint subsets. In particular, fuzzy logic is used to distinguish three subsets of configurations as *small resources wastage*, *low power* and *low temperature*. A "membership function" assigns a value per candidate, as for the degree it satisfies each configuration subset. During the evaluation phase of the GGA, the membership functions per candidate are assessed. The algorithm selects solutions with the highest degree of membership to all groups.

More recently Frey et al. [74] encode the problem of optimising the deployment of software in the cloud as a search-based problem. The authors are concerned with the deployment of software to cloud VMs but not with the mapping the configured VMs to the available PMs. Each deployment decision - so called a cloud deployment decision (CDO) comprises a combination of a specific cloud environment, services composition and reconfiguration rules. The considered reconfiguration rules are described as "shrink", that removes VMs from the current CDO, and "grow" that adds new VM instances to the current CDOs. The design space of all possible CDOs is explored with the NSGA-II algorithm [57]. The CDOs that best balance the trade-offs between configuration cost, response time and sla violations are finally selected. Due to the long convergence time of the algorithm, the authors restrict the population size of the algorithm to 50 individuals.

Similar with Frey et al., Zhao et al. employ the NSGA-II genetic algorithm to optimise the scaling of VMs in dynamic cloud scenarios to meet SLAs [212]. The authors aim to develop an adaptive VM scaling framework that satisfies the changing user requirements at a minimum cost. Each VM is encoded as a chromosome, and

a group of chromosomes represent a possible scaling solution. The quality of each solution is estimated by running a k-nearest neighbour algorithm to estimate the response time of the VM configuration. Based on the estimation, the SLA satisfaction ratio is measures and the most promising configurations are selected accordingly.

Xu et al. propose a framework for the dynamic deployment of VMs in cloud datacenters towards optimally managing the users' requirements at runtime [203]. The optimisation criterion is defined here as the maximisation of resources' usage. Each PM's resources are modelled as a triplet of available CPU units, memory size and bandwidth. Similarly each VM is a triplet of requests CPU, memory and bandwidth. A parameter matrix H is defined that describes the deployment options for all VMs to the available PMs. The authors resort to particle swarm optimisation evolutionary heuristic search the matrix H for the optimal parameters. Candidate solutions are evaluated using an a priori defined analytical objective function, that models the optimisation objective of the system i.e., maximum resources usage. The authors terminate the algorithm when 500 iterations are reached, that results in a fast convergence time of approximately 20 seconds.

Guo et al. [84] combine machine learning and genetic algorithms to flexibly provision cloud resources to multi-tier applications. In particular the authors focus on provisioning the optimal cache size to current workloads to improve the request processing times. First, a clustering algorithm is employed to classify the incoming requests as static or dynamic. According to the identified request content a genetic algorithm is used to search different caching combinations that will minimise the total processing times of all requests. Candidate caching solutions are evaluated using an a priori defined analytical formula to estimate their processing costs. Although the presented approach is an interesting solution to dynamically provision infrastructure resources to applications, the authors introduce a predefined at design time fitness function to assess the solutions' quality. It is not detailed how the formula is derived

but if design-time assumptions do not hold, the achieved configurations may become inefficient, systems may saturate, and services may stop behaving as prescribed by their SLAs.

More recently Pandit et al. [158] have proposed a solution to maximise resources utilisation in cloud environments and therefore offer cheaper configurations to cloud users. The authors formulate a multi-parameter bin-packing problem to allocate VMs to PMs. Each parameter represents a computing resource type i.e., CPU, RAM, storage and bandwidth. Then the simulated annealing evolutionary algorithm is employed to search for the most cost-efficient resource configurations [143]. Similar to Guo et al. [84] the authors also apply an analytic formula to evaluate the cost of candidate solutions, that may as already discussed restrict the runtime applicability of the framework.

Casalicchio et al. [38] address the issue of optimally allocating VMs to PMs in clouds, to maximise the infrastructure provider's revenue. The authors formally specify the provider's revenue as a function of the total VM migrations and the penalty to outsource VMs to another cloud provider in case the workload demand exceeds the maximum resource availability of the system. The authors resort to local search, proposing a heuristic based on hill climbing techniques [38], to search for VM allocations that optimise the provider's revenue.

Barrett et al. apply reinforcement learning techniques to tackle the problem of dynamically scaling the cloud VM configurations to meet varying application workloads [17]. Reinforcement learning techniques operate on the premise of punishment and reward, with agents biased towards actions that yield the highest rewards. The reinforcement learning problem is modelled using markov decision processes (MDP), a mathematical framework to model decision making under uncertainty. A MDP is represented as a four tuple consisting of system states, actions, transition probabilities and rewards. The cloud system state is modelled as the conjunction of three

variables; the total number of user requests, the total number of allocated VMs and timestamps. At each time step the agent chooses among all possible actions within the current state, those leading to the highest reward. Rewards are estimated via an analytic formula based on a pre-defined SLA denoting the applications' performance. To improve the convergence time of the algorithm, the authors introduce a parallel reinforcement learning solution, where neighbouring agents operate simultaneously on the same task. However neither the initial nor improved execution times are discussed in this work. Similar reinforcement learning based approaches to optimise cloud configurations are presented by Dutreilh et al. [61] and Rao et al. [164].

Summarising the aforementioned approaches of Guo et al. [84], Pandit et al. [158], Casalicchio et al. [38], Xu et al.[203], Zhao et al.[212], Frey et al. [74], Zeng and Ye [205] we observe that the application of evolutionary metaheuristics to extract near-optimal solutions is growing in popularity in the cloud literature. However the computational complexity of evolutionary algorithms highly depends on the complexity of the fitness function used to assess the quality of candidate solutions. It is worth noting that a computationally cheap a-priori known analytical fitness function as used e.g. in [158, 84] may introduce design time assumptions, that can hamper the generalisation ability of the framework. On the other hand a highly accurate fitness function derived from simulation or physical experiments as e.g. in [74] can introduce prohibitive convergence times. Existing contributions employing evolutionary metaheuristics focus on the exploration of near-optimal configurations rather than on the challenge of runtime adaptations.

2.3 Proactive Methods

The goal of prediction-based methods is to forecast possible changes in the environment such as workload fluctuations or changing resource usage patterns. By antici-

pating future changes, the system aims to *proactively* adapt its configuration to avoid performance problems or inefficient resource usage.

Wood et al. have proposed "Sandpiper", a framework for proactive SLA management in virtualised environments [201]. An SLA violation occurs if the aggregate usage of any resource (i.e., CPU, memory or network) exceeds a predefined threshold W. The authors first use OS-level statistics to construct resource usage profiles for each VM. Future resource usage values are predicted using time series analysis, based on the collected profiles. In particular the framework relies on first order autoregressive predictors [28] for predictions. Once a hotspot has been detected Sandpiper decides if it can be resolved locally within the VM or needs migrations. In the first case, VM resizing actions are employed to align the current VM capabilities with the actual workload. Otherwise, if there are sufficient free resources on other PMs, a greedy heuristic is triggered. PMs and VMs are sorted in a decreasing order by their degree of overload. The algorithm proceeds by placing the highest loaded virtual machine from the highest loaded server to the least loaded physical server. If there is not enough free space in the PM hosts, VM swaps are triggered. A swap involves exchanging a high load virtual machine hosted in a highly loaded PM with one or more low loaded VMs from an under-loaded PM.

The use of black-box modeling for workload prediction is appropriate for cloud scenarios since it enables loose coupling between the infrastructure and services layers. The authors have focused on an approach that takes corrective action as a response to workload fluctuations. However the approach does not support the identification of optimal or near-optimal configuration solutions to balance non-functional trade-offs.

A similar approach is proposed by Bobroff et al. [27], towards minimising the overall cost of running a virtualised datacenter. In other words the authors propose a framework to dynamically manage SLAs while minimising the number of active hardware servers. Honouring SLAs is translated to providing guarantees for sufficient

CPU access to VMs. Based on this, the goal is to minimize the number of active PMs, subject to the constraint that the probability of exceeding the VMs' actual capacity does not violate a predefined threshold p during datacenter operation. The architecture of the proposed system is comprised of a monitoring unit, a time series forecasting module and a resources reconfiguration module. First, the resource demand of each VM is forecast based on the monitored data. The prediction is fed to a greedy first-fit heuristic bin-packing algorithm, that maps the available VMs to PMs such that the overloading probability of each VM will be less than p. The reconfiguration algorithm is triggered periodically, every 15 minutes.

Time-series prediction is a powerful tool for resource provisioning if the system's parameters exhibit regular patterns. Either off-line or on-line training can be used to capture the system's dynamics and provide accurate predictions. However, the technique is not general and heavily depends on predictable system patterns.

Kusic et al. [126] present a methodology towards the maximisation of "revenue" in virtualised datacenters. In other words, the authors' goal is to minimise the power consumption and the penalties imposed due to SLA violations, towards reducing the overall expenditures in the system. Additional costs, so called "costs of control" are assigned to the resource reconfiguration actions. Such costs of control include the power consumed by machines being powered up or down during reconfigurations, without performing useful work. The cost of migrations is not considered in this study. The problem of minimising the total power consumption, SLA violations and control action costs is expressed as a utility maximisation problem, solved with lookahead control [5]. Future workloads are predicted using a Kalman filter. The a priori knowledge on workload arrivals, is used to estimate a sequence of control actions over a prediction horizon to improve the expected utility. The possible control actions may be resize of the VMs' CPU, as well as tuning of the number of active VMs and PMs. At each time step the lookahead controller applies only the first action of the

sequence and the rest are discarded. The entire process is repeated at the next time interval, given new environment information and updated workload forecasts. The framework may operate on a small datacenter, with up to 5 PMs.

The approach assumes a-priori known workload patterns. However previously unseen utilisation patterns may cause the predictive controller to fail to provide adequate reconfiguration actions.

Chaisiri et al. [39] propose a methodology for optimal VMs allocation across multiple cloud infrastructure providers. The authors address the challenge of minimising the users' costs, for reserving an optimal amount of VM instances to host their application workloads. The proposed framework distinguishes three operational phases; reservation, utilisation and on-demand. In the reservation phase the cloud broker provisions an initial resources plan to the users. The utilisation phase starts when the reserved resources start being used. Finally the on-demand plan provisions possible additional resource requirements, that cannot be covered by the initial reservation plan. The application workloads and reservation prices are considered a priori known, based on statistical analysis of historical data. Then, the authors obtain the optimal solution by formulating a deterministic integer programming problem, to minimising the total cost of VMs reservation based on the user VM instances requirements and the provider prices. In case that new trends are emerging in the cloud and the history data are no longer relevant, the uncertainty in the prices and user demands is expressed via a stochastic integer program formulation. An exact solution is obtained by transforming the stochastic program to a deterministic equivalent.

The authors do not detail the convergence time of the algorithm and how such resource reservations can be used to balance the trade-off between infrastructure costs and application performance.

Chuen et al. [50] propose another proactive solution to the problem of optimal VMs placement in cloud datacenters. First, the application workloads are predicted

using a combination of different techniques namely kalman filter, exponential smoothing and markov chains. Then, to balance the trade-offs between the cloud provider's resources capacity, the cost of the total configuration and the users' demand the authors resort to genetic algorithms. The authors apply a hybridized algorithm that combines particle swarm optimisation and ant colony optimisation [143]. To avoid traps to local optima the algorithm tracks at each iteration the current global best solution, and propagates it to the next iterations. The "goodness" of each candidate, is evaluated using a mathematical formula that assigns a cost to each configuration, based on predefined resource reservation prices. Ant colony optimisation was also used by Li et al. [133] to load balance workload requests to the cloud VMs, aiming to minimise the total response time of the workload. The approach clearly improved the response times, in comparison with greedy bin packing heuristics.

Feller et al. have proposed a consolidation approach for energy efficient cloud configurations. Workload demands are assumed a priori known [66]. A workload is considered request for VMs. The problem is reduced to an instance of Multi-dimensional bin packing problem, in which the PMs represent the multi-dimensional bins of resources (CPU, network bandwidth, RAM) and the workload are the items to be packed in the bins. The authors employ evolutionary algorithms, i.e., ant colony optimisation to minimise the number of active servers and therefore consolidate the resources pool. The validation results compare the proposed approach with bin-packing heuristics. The results reveal marginal energy savings and significantly higher computation times. This shortcoming may be related with the constructive-based approach of ant colony optimisation, where possible configurations are constructed and explored probabilistically. The probability values based on which the search space is explored, are manually tuned which might be a hindrance to fast exploration. In a later work, the authors resort to a series of greedy bin packing heuristics to achieve energy efficient cloud configurations [67].

The constructive approach of ant colony optimisation heuristics as used in the aforementioned [50, 133, ?] does not seem well applicable in the cloud reconfiguration problem space. More specifically ant colony optimisation employs the notion of "pheromone trails" construction to indicate the desirability/fitness of solution components. The pheromone trails have to bee adjusted with time to update fitness of solution according to newly discovered components [143]. We observe that the authors in [50] have deduced the pheromone modeling process to a predefined mathematical formula. The authors do not detail how the mathematical has been derived and how this may be generalised to other cloud environments.

A solution to the problem of maximising the provider's revenue is proposed by Wu et al. [202]. The optimisation objective is to minimise the cost of using the cloud by minimising the number of active VMs and to reduce the penalties of SLA violations. The authors observe that the cloud industrial services typically allocate dedicated VMs to each customer that might end up wasting hardware resources due to under-utilisation at non-peak loads. Based on this, Wu et al. differentiate their approach by consolidating many user requests to a single VM. An additional heuristic is employed to minimise possible SLA violations resulting from the consolidation by reserving resources according to the customers' future resource requirements, estimated according to historical records.

Huber et al. propose another solution to address the performance versus efficient resources utilisation trade-off [99]; highly variable workloads make it challenging to provide quality-of-service guarantees while at the same time ensuring efficient resource utilisation. The authors use the palladio component model [20] as an architecture-level performance model to explicitly model usage profiles and resource allocations. Predictions are achieved by simulating a queueing network. After the future usage patterns are extracted a greedy heuristic is applied to optimise the resources usage. If SLA violations are identified, the "push" phase of the algorithm increases the amount

of allocated resources of all types (i.e., CPU, memory). To improve the resource usage the "pull" phase of the algorithm releases resources that are not utilised by current applications. If an SLA violation is predicted as a result of the change in resources, then the change is reversed.

The approach does not provide means to reason on the effect of uncontrolled changes in working conditions in cloud environments, that may affect the availability of physical resources and the end-to-end QoS of applications that they run.

More recently Gambi et al. [76, 75] have proposed the use of kriging models (i.e., Gaussian processes) to predict and manage service performance under different workloads. Kriging models offer a black-box method to model workloads as opposed to analytical models as in [99]. Black-box models offer an advantage in the context of cloud over white-box models because the IaaS and SaaS providers are often different organisations and therefore IaaS providers may only access limited information about the services due to legal boundaries [149]. Additionally clouds are highly dynamic therefore the controllers must be quickly to learn and fast to adapt to the fast changing workloads conditions. As the authors demonstrate, kriging can be tuned at runtime, that is an advantage over analytic or rule based models that must be tuned at design time. After the performance has been predicted, a generic decision maker decides the most relevant corrective action.

Gambi et al. have contributed a methodology for predictive workload modeling in cloud environments. However the approach does not offer an optimisation solution to exploit the results the predictive analysis.

Jung and Kim [115] are concerned with the problem of optimising the deployment of service workflows in heterogenous clouds towards reducing the cost of executing services and optimising the services' execution times. First the authors use a directed graph to formalise the search space of all possible deployments. Next time series analysis and in particular the auto-regressive moving average filter is applied to predict

the computation times per workflow. The optimisation problem is then reduced to a weighted shortest path problem and solved with a heuristic search algorithm that iteratively revisits alternatives to find better paths. To reduce the running time of the algorithm the authors reduce the graph vertices based on a criticality criterion.

The authors map the initial problem to a simplified equivalent, but do not assess the trade-off between the timeliness of the optimisation process versus the optimality of the achieved solutions.

Fernandez et al. [69] observe that the majority of current cloud scaling platforms offer the same QoS level to all customers, despite the fact that different customers typically have varying preferences e.g., different requirements for service availability and performance. To fill this gap, the authors propose a resources scaling framework that adapts to changing workloads considering the different customer QoS requirements. In the proposed scheme each customer can tune its own cost/SLA fulfilment trade-off using a "metal" classification scheme (i.e., gold, silver and bronze customers) which defines different criteria for the selection of scaling plans based on the respective customers' QoS preferences. To prevent SLA violations in advance, time-series analysis is first used to predict the future services demand, based on past resource usage monitoring data. Horizontal scaling reconfiguration actions (i.e., the possibility to add and remove VM instances) may be explored to adapt the system to workload variations. To decide which type and amount of VM instances must be added or released, a decision tree is constructed, that contains all possible reconfiguration combinations. The cost of each scaling plan is estimated as the cost of the infrastructure usage and the cost incurred by possible SLA breaches. For each customer the optimal performance/cost path is different according to his metal classification; i.e gold users pay more to get a higher QoS while bronze users are willing to accept QoS degradations for a cheaper configuration.

Fernandez et al. have assumed the existence of a cloud resource reconfiguration engine. On top of the later, they have contributed a pricing model to harness varying client budgets.

Zhang et al. [210] are motivated by enterprise IaaS providers and observe that the price of the VM instances fluctuates based on the rule of supply and demand; when the demand is low, the datacenter becomes underutilised and renting VMs is cheap. On the other hand when the demand increases it is desirable for the cloud provider to raise the price of the instances to increase revenue. The authors propose a framework to maximise the total IaaS provider's revenue while minimising the energy costs. The authors resort to control theory to formalise the problem of dynamic resource allocation. The future customer demand is modelled using time series analysis and the model is used to exercise control on the cloud resources configuration over a time horizon. Each VM is modelled as a M/G/c queue. Changes in the configuration are associated with a penalty. The goal is to minimise a cost function that expresses the cost of configuration penalties and price changes over the time period of control. The problem is finally solved analytically by exhaustively exploring a series of possible reconfiguration actions in the control period.

The authors provide useful insights on connecting the infrastructure usage with pricing estimates. However the application of M/G/c queues to model the cloud VMs is not generalisable since explicit assumptions are made on request arrival times and services' runtimes.

2.4 Conclusions

Cloud computing extends traditional virtualised systems with the requirement for elasticity [95] i.e., the requirement of cloud datacenters to automatically adapt to environmental changes to match the current demand as closely as possible. Elasticity

has two dimensions; *speed* and *precision* [95]. Therefore resource reconfigurations must not only be fast towards meeting dynamically changing workloads but also precise to achieve maximum operating cost savings such as energy consumption costs. Dynamic environments may lead to complete rethinking of the best strategy to improve configurations. For example when the workload is rapidly changing, it may be better to suffer a slight performance degradation rather than trigger expensive reconfigurations whose costs may never be recouped before another adaptation is needed [114]

In this chapter we have presented an overview of current cloud resource provisioning approaches. Resource provisioning frameworks manage the datacenter configurations to guarantee satisfactory performance under changing conditions. Therefore resource provisioning methods are key facilitators of an elastic design. The most common optimisation goals in the literature are the maximisation of services performance (typically measured in SLA violations) and the minimisation of energy consumption costs. These optimisation goals are in mutual conflict with each other, therefore a solution that combines a globally optimum energy consumption and a globally optimum performance is not realistic. Instead, the literature focuses on achieving a good balance between these trade-offs.

Existing commercial IaaS clouds such as Amazon EC2 rely on the users to specify the conditions for adding or removing servers according to their changing needs. While such rule-based approaches are easy to implement, they are imprecise and wasteful in terms of resources usage and SLAs fulfilment [69]. Consequently more potent academic solutions have been proposed. We have divided the literature overview in two main categories based on the way they handle the dynamic cloud environment changes such as e.g. workload fluctuations; *proactive* and *reactive*. Proactive schemes resort to workload predictions to pro-actively plan an optimal amount of resources and guarantee SLAs. When the predictions are accurate, this scheme provide very good

results. However predictions can be inaccurate when the changes are unpredicted. With reactive schemes the resource allocation is adjusted on-demand based solely on recent behaviour;a change is detected and the allocations are adjusted accordingly. Reactive allocation is computationally attractive since it does not require extensive knowledge of the services resource demands. Additionally legal boundaries between cloud vendors, further restrict the access on service's knowledge. However, the reactive schemes' efficiency in practice depends on their ability to adjust allocations in response to changes in a timely fashion [117].

Apart from the use of predictive modelling techniques, both reactive and proactive contributions resort to similar methods to optimise the resource configurations. The optimisation techniques listed in Sections 2.2 and 2.3 vary from simplistic to very sophisticated contributions. Frameworks that attempt to identify a global optimum for performance or costs, typically specify a constraint, single objective optimisation and employ *exhaustive* optimisation techniques to solve it [70, 96, 31, 9, 159, 62, 62, 17, 126, 39, 115, 69]. The discussed exhaustive optimisation techniques include control theoretic formulations, integer programming, decision trees and reinforcement learning approaches. These methods search exhaustively the cloud design space for the best configurations actions and offer the advantage of high accuracy. However cloud design space is large due to the number of the available configuration parameters. Each machine contains a large number of configurable parameters, such as CPU time, memory, and network bandwidth, and the VMs on the same host may interfere with each other. Therefore the time that exhaustive optimisation algorithms take to converge conflicts with the requirement of rapid elasticity. This is possibly the reason that analytic methods are typically evaluated on trivial cloud systems including a minimum number of PMs and VMs (e.g., 6 nodes in total [126]). Additionally techniques as integer programming or control theory make strong assumptions about

the cloud environment such as linearity that are seldom verified in highly varying environments [77].

Heuristic algorithms are implemented by the majority of of the proposed contributions to balance accuracy and speed of convergence. Simpler implementations include bin-packing heuristics that greedily pack VMs to PMs [83, 106, 132, 201, 27, 67] to either avoid SLA violations or increase resources usage. We observe that greedy algorithms can be fast to execute but often lead to low quality approximations of near-optimal configuration solutions. An example of a problematic situation that may be caused by greedy heuristics is system instability; greedily switching-on servers to avoid SLA violations can lead to violations of allowed energy consumption thresholds. In a similar fashion, servers will be greedily switched-off to resolve the energy consumption thresholds violation, possibly triggering performance degradation of hosted services, that will call again for activating new servers etc.

Stochastic search algorithms are a powerful tool to explore near optimal configurations when the configuration space is huge and an exhaustive search is infeasible [89]. Multi-Objective Evolutionary Algorithms (MOEAs) is the most popular technique used in the current state-of-the-art to efficiently explore large design spaces balancing the trade-offs of *exploration* of new areas of the design space with *exploitation* of already known good solutions. Discussed evolutionary meta-heuristics in this chapter include ant colony optimisation [66, 67], NSGA-II [74, 212] and particle swarm optimisation [203]. However, MOEAs' complexity highly depends on the complexity of the function used to evaluate the quality of candidate solutions, so called fitness function [86]. In the datacenter configuration optimisation problem, there is no obvious analytical expression for estimating the quality metrics of interest for a candidate datacenter configuration. Highly time-consuming simulations or physical experiments are typically employed to evaluate configurations, leading to cost prohibitive optimisation processes, requiring hours or even days to converge [74].

We observe that existing frameworks do not systematically consider the trade-offs between optimality and cost of the optimisation process limiting their applicability to offline scenarios. The convergence times of the proposed frameworks are not reported with the exception of Xu et al. in [203]. In the remainder of this thesis, we develop a methodology to optimise datacenter configurations at runtime. Contrary to existing approaches, we will also explicitly consider the optimisation time costs.

Chapter 3

Research Scope

The literature surveyed in Section 2 uncovered a number of existing approaches on cloud datacenters resource provisioning. A distinction was made between proactive and reactive control; proactive approaches use predictive demand to allocate resources before they are needed while reactive schemes react to immediate fluctuations before periodic demand prediction is available [209]. In this chapter our goal is to scope our research.

In particular this Chapter is structured as follows: In Section 3.1 we analyse the problems of over-provisioning and under-provisioning in enterprise datacenters. In Section 3.2 we present an industrial case study to examine these limitations in practice and convey the challenges of purely proactive resource management methods. Next, in section 3.3 we present our motivating scenario focusing on the challenges of cloud datacenters resource provisioning that this thesis will aim to address. Next, in Section 3.4 we formulate our problem statement. Finally in Section 3.6 we describe how we model the cloud environments under study using simulation.

3.1 Over-provisioned and Under-provisioned Datacenters.

Changes in the environment conditions may cause the datacenter to transit from a stable to an unstable state where functional and/or non-functional targets are missed. An example of environment changes is varying usage patterns (e.g. seasonal customer demand spikes such as during Christmas shopping). Workload spikes may stress the system more than its provisioned capacity and therefore compromise non-functional quality attributes (such as e.g., response time and reliability). [124].

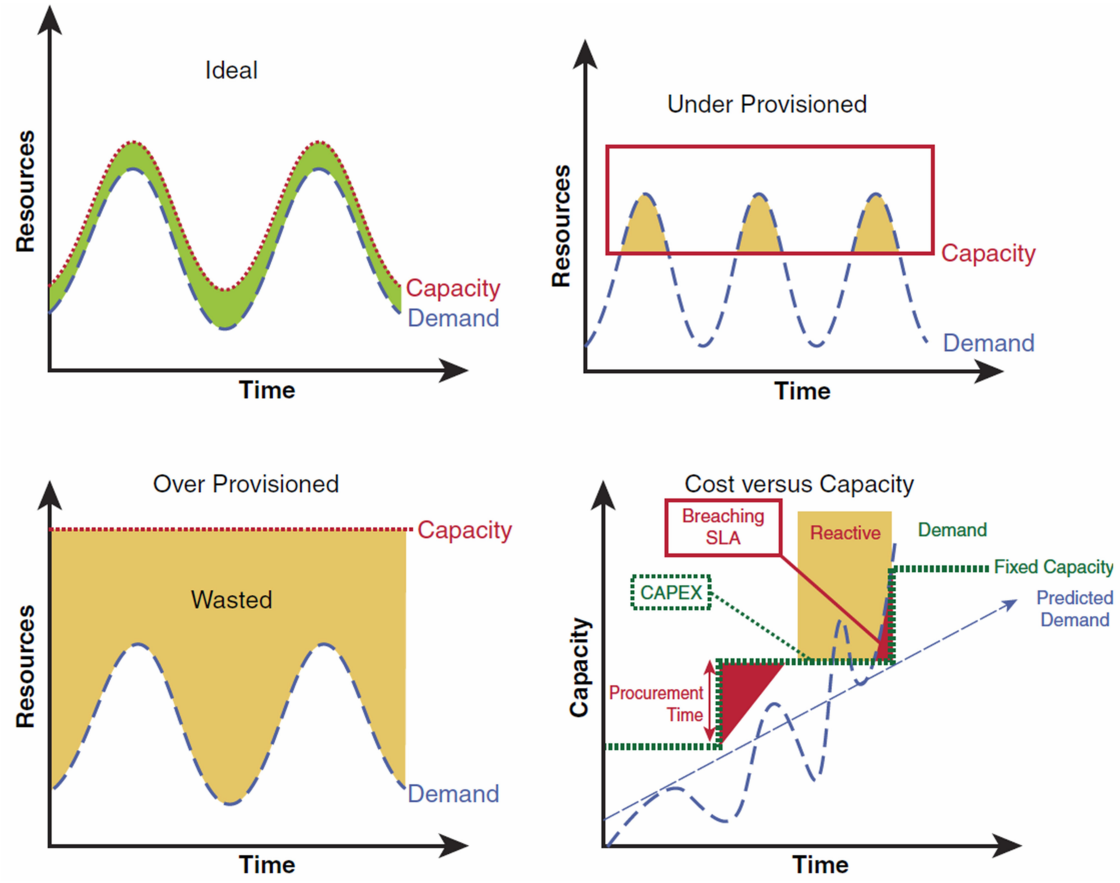


Figure 3.1: The effect of elastic adaptation [113].

As Figure 3.1 illustrates, an ideally elastic system dynamically self-optimises its available resources to closely match the actual demand. If the expected demand esti-

mation is set too low, the system is *under-provisioned* and peaks of resource requirements cannot be met. This results in performance degradations are measured e.g. with SLA breaches and consequently poor customer experience. On the other hand if the provisioned resources exceed the actual demand the system is *over-provisioned*, which causes significant capital and operating costs to the system providers. Over-provisioning is a common tactic in modern datacenters to guarantee adequate performance and availability at peak usage times [14]. Currently the average server utilization in a typical datacenter does not exceed 30%, but idle servers still consume 60% of peak power [146]. Therefore, over-provisioning results in the proliferation of enterprises' operating costs [146].

The requirement of engineering elastic datacenters involves the datacenters' ability to anticipate and optimally adapt to dynamic environment changes by continuously guaranteeing the system's QoS performance goals. A traditional approach to tackle this challenge is *proactive* resource allocation which employs predictive models to estimate in advance the datacenter's future capacity requirements. Existing methodologies often build architecture-level detailed models that explicitly capture the software architecture and the execution environment of the system. The goal of these efforts is to predict the system performance by transforming architecture-level performance models into predictive performance models (e.g. Layered Queueing Networks or Petri Nets) [100]. Another important tool consist trace-based methods such as time-series analysis [150]. Here the general idea is that historic traces offer a model of the environment, which will be representative of the future applications and user behaviour.

As a result of a proactive resource management the physical resources are already installed and changes are managed seamlessly with zero procurement time [117]. When predictions are accurate this scheme provides very good performance. However it fails to accommodate newly seen trends such as when unprecedented sharp changes

in workloads occur. Proactive methods also fail when the collected history traces do not follow any predictable patterns or in the case of very noisy datasets[117]. Another disadvantage of the proactive approach is that the models are build at design time to estimate the future resource needs of the running system. However, at system design-time the model parameters are often estimated based on approximation techniques which may hamper the accuracy of the predictions [32]. In Figure **??** the graph "Cost versus Capacity" highlights with red the inaccuracies between the predicted system capacity and the actual resource demand. Any failure to accommodate the additional system demand results in SLA breaches which translate to monetary penalties and reputation decline of the system provider [163]. Furthermore inaccurate resources procurement cancels the cloud vision, which promises an infinite set of resources to the cloud consumer through its ability to elastically adapt to environment changes.

3.2 An Industrial Workload Study

During my internship in "CAS Software AG" [1] from March to May 2014 I was given the opportunity to study enterprise workloads and assist the engineers with the following tasks: (i) characterisation of workload patterns (ii) identification of possible significant changes in user behaviour and (iii) assessment of the resource pool capacity needed to support the workloads. Based on the above, we will examine in this section industrial resource utilisation patterns and explore the notions of over and under-provisioning in practice. Based on the observed utilisation patterns we will employ a predictive resource provisioning method towards better matching available resources to incoming workloads.

"CAS Software AG" is a small and medium-sized enterprise (SME) offering to other SMEs customer relations management (CRM) solutions. We have analysed

[1] `http://www.cas.de/en.html`

workloads from "CAS PIA" [2], a CRM tool offering functions as appointments management, project management, e-mail integration and mobile synchronisation. "CAS PIA" features an industry standard 3−tier architecture.

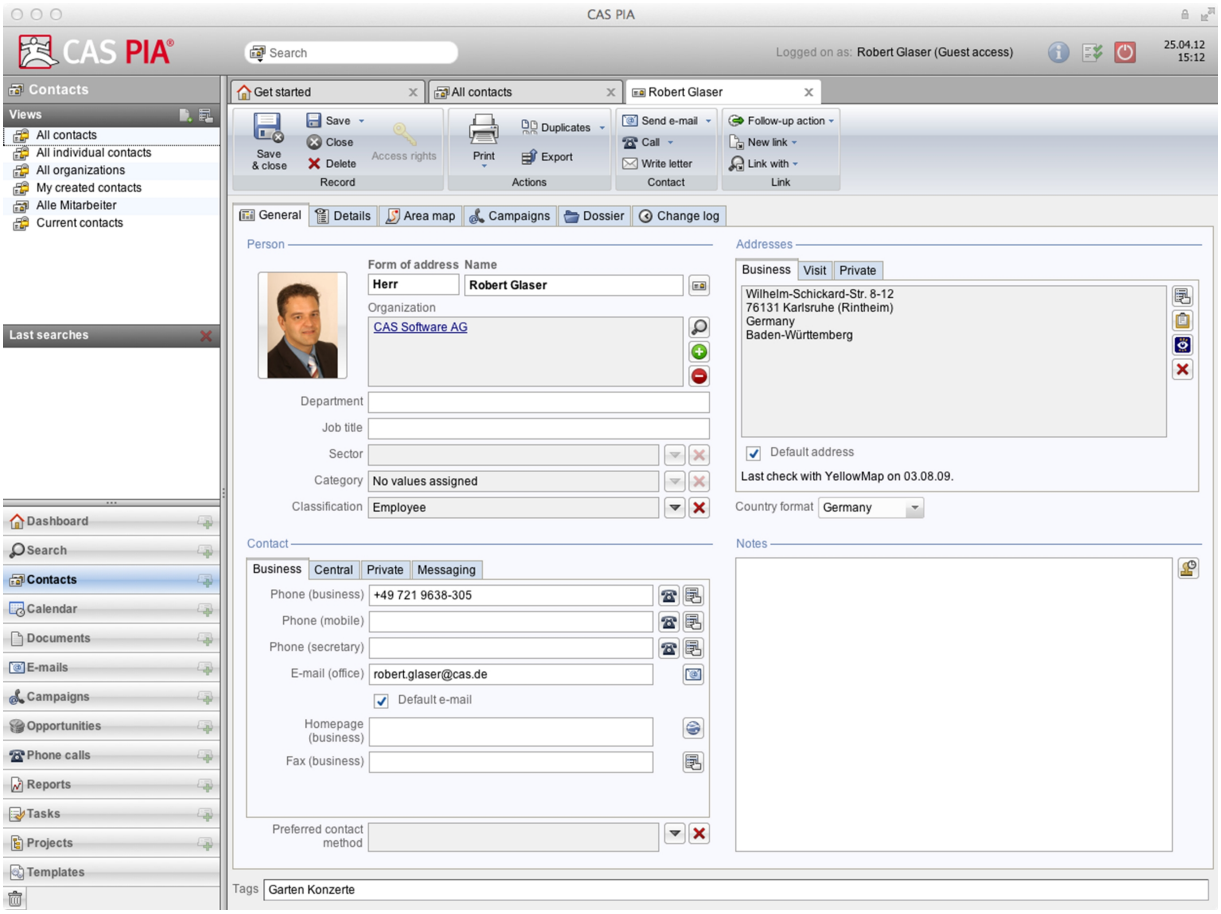


Figure 3.2: The CAS PIA user interface with mock data.

The data were collected from the application tier during the 15 day period 7-21 April 2014 and describe transaction arrivals and CPU utilisation records. Days $12-13$ and $19-20$ correspond to weekends. Days $18-21$ correspond to bank holidays. A transaction is defined here as a user activity in the "CAS PIA" console, shown in Figure 3.2. There exist 10 unique system-level transactions, that originate from user interface-level transactions such as e.g., pressing the "save" button, summarised in the first column of Table 3.1. The second column of Table 3.1 indicates the relative

[2] `http://www.cas-pia.com/`

frequency each transaction in the monitored total of $8,978,568$ transaction events. Overall two data traces were devised for the analysis; one containing transaction arrivals per hour and one containing CPU utilisation (%) per minute. The two data traces will be studied in parallel as they convey complementary information.

It is important to understand the past when trying to anticipate future developments. Therefore it is useful to extract deterministic patterns that occur in time series history. Successful forecasting of patterns could both alert operators of incoming crises and also enable an optimal proactive allocation of CPU resources. To this end we have used time series analysis to predict upcoming periods of high utilisation. All discussed models have been implemented using the R forecast package [101]. A time series X is a discrete function that represents real valued measurements $x_i \in \mathbb{R}$ for every time point t_i in a set of n equidistant time points $t = t1, t2, \ldots, tn$: $X = x1, x2, \ldots, xn$ [179]. A time series can be decomposed into the three following components [28]: (i) *trend:* which describes a monitoring increasing or decreasing function (ii) *season:* which captures recurring patterns in the data and (iii) *noise:* which is an unpredictable, random overlay of frequencies.

Time series analysis offers a broad spectrum of methods to calculate forecasts based on monitored history data [94]. One of the most generic and widely used [211, 27] is the Auto-Regressive, Integrated, Moving Average ARIMA(p,d,q) model [28].

Transaction Type	**Frequency**
createObject	0.00056%
echo	0.48%
execute	1.83%
getAvailableObjectNames	0.44%
getObject	0.75%
getObjectDescription	2.87%
query	94%
saveAndReturnObject	0.00055%
saveMultipleObjects	0.0002%
saveObject	0.002%

Table 3.1: CAS PIA transaction types and frequencies.

The three model parameters are the *auto-regressive (p)*, *integrated (d)* and *moving average (q)*. The p autoregressive terms describe the dependencies among successive observations. The integrated element d suggests trends in the data. Finally the q moving average terms describe the persistence of random error from one observation to the next. Using ARIMA, the underlying process that generate the time series has the form:

$$Y_t = \phi_1 Y_{t-1} + \cdots + \phi_p Y_{t-p} + \epsilon_t + \theta_1 \epsilon_{t-1} + \cdots + \theta_q \epsilon_{t-q} \tag{3.1}$$

where Y_t and ϵ_t correspond to previous observations and error accordingly while $\phi_p \neq 0$, $\theta_q \neq 0$.

The parameters of the ARIMA models are identified by examining patterns in the plots of the autocorrelation function (ACF) [175] and partial autocorrelation function (PACF) [175] of the time series[179]. ACF measures the linear predictability of the series at time t, while PACF measures the ACF between Y_t and Y_{t+k} time series observations with the linear dependence through the observations removed. Figure 3.3 shows the ACF and Figure 3.4 the PACF for the transaction arrivals time series. The observed sine-wave in the ACF plot indicates the existence of trend in the time series. The ACF plot shows three spikes in the first three lags which indicate the existence of $q = 3$ moving average terms. The PACF plot spikes at lag two, which indicates the existence of $p = 2$ autoregressive terms. We further confirm the identified parameters using the R *auto.arima()* function. We therefore devise an ARIMA(2,1,3) model to predict the future transaction arrivals. The forecast and the observation values in the course of time are plotted in Figure 3.5.

We observe that the periodicity and average values of the time series are appropriately modelled by the ARIMA model. However the majority of arrival peaks are not correctly anticipated. To avoid dropping incoming requests, the widely followed approach of over-provisioning resources employs a dedicated group of servers for each application with enough total capacity to accommodate requests at peak rates [117].

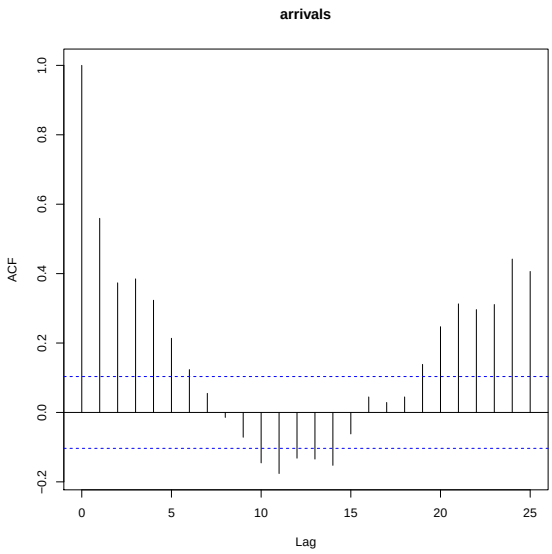


Figure 3.3: ACF for transaction arrivals.

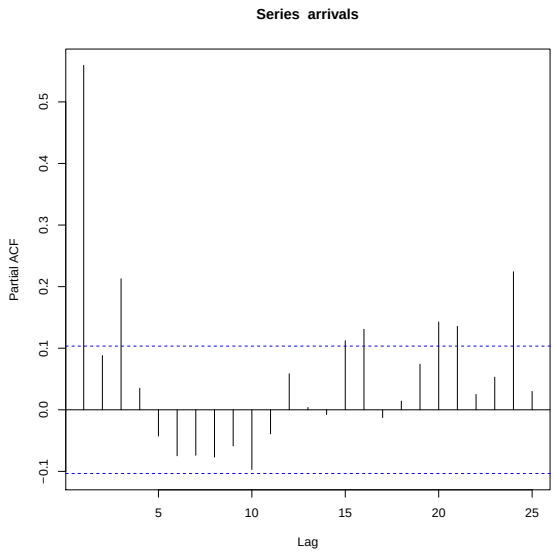


Figure 3.4: PACF for transaction arrivals.

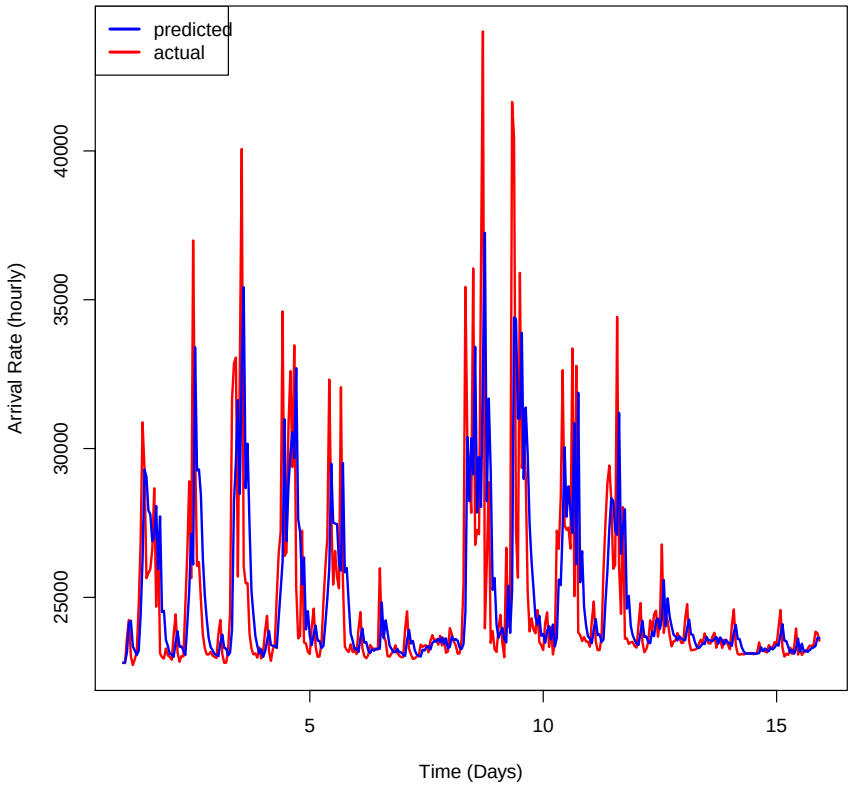


Figure 3.5: Comparison of the actual and predicted observations of the transaction arrivals time series.

The downside of this tactic has been a server and storage sprawl with low utilization rates [129]. The same practice of over-provisioned hardware can be clearly depicted in the analysis of the CAS CPU utilisation traces.

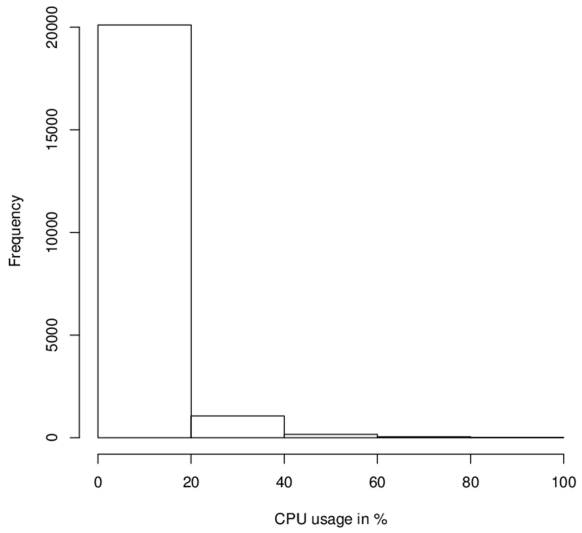


Figure 3.6: Histogram of CAS CPU utilisation. We observe that mainly CPU usage falls within 20%.

Figure 3.6 shows that CPU utilisation mainly oscillates within $0-20\%$, suggesting that the CAS hardware servers are significantly over-provisioned. Figure 3.7 shows the raw CPU utilisation time series. In comparison to the transaction arrivals time series, the CPU time series is not suitable for proactive analysis. We observe a lack of a distinct periodic behaviour while the ACF in Figure 3.8 is weak and quickly decaying. The CPU trace characteristics make it difficult to forecast future utilisation patterns [27].

Summary and Vision

Resource provisioning is central for every application to comply with its SLAs. The datacenter capital and operating costs are directly related to the selected resource provisioning techniques. Common practices such as machine dedication to applications [13] and over-provisioning have caused significant financial implications to

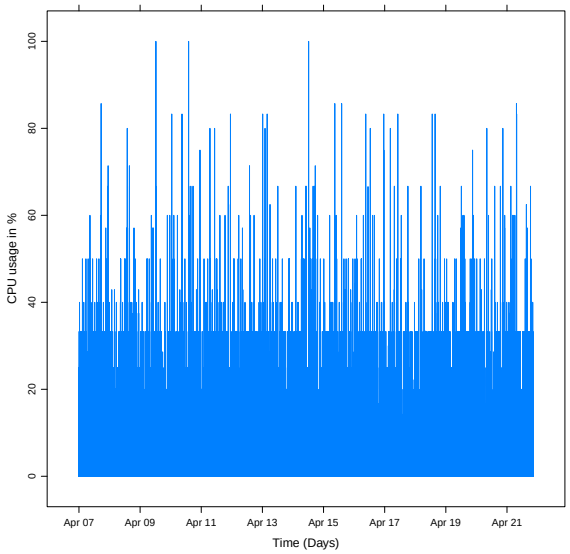


Figure 3.7: CPU utilisation time series.

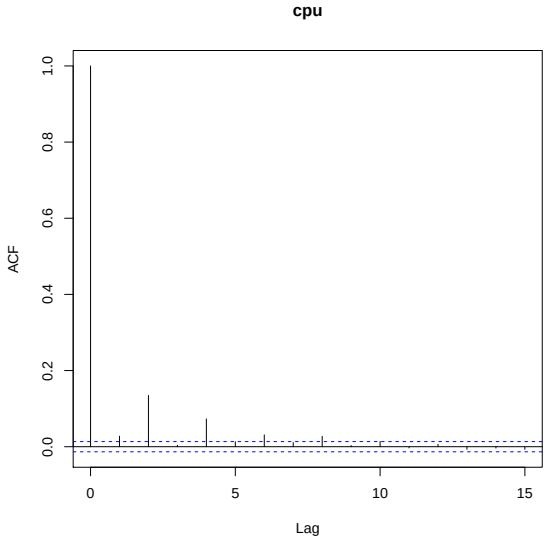


Figure 3.8: ACF for the CPU time series.

enterprises, stemming from the ever increasing data centre size required to host more applications that grow as well in size and complexity [117]. Our findings in Section 3.2 are consistent with several reports, demonstrating that hardware servers remain underutilised most of the time [146]. As quoted in [117] *"According to Tony Iams, Senior Analyst at D.H. Brown Associates Inc. in Port Chester, NY, servers typically run at* $15-20\%$ *of their capacity"*. In addition, it is reported that that current enterprises have spent 140 billion dollars more in capital infrastructure expenses than necessary to satisfy their workload requirements [3]. Keeping large scale computing facilities and datacenters running, require large amounts of energy, that results in proliferation of operating costs. For example in 2006 datacenters consumed 28 billion kWh, equalling approximately 2% of the total U.S electricity consumption and trends show that datacenter power consumption keeps growing by 18% annually [132, 162]. In 2015 the estimated energy consumption of US datacenteres reaches 100 billion kWh [24].

To alleviate these issues datacenters must have fewer but more utilised machines. Cloud Computing is a paradigm where optimisation of resources usage and services

performance lies in the very center. Elasticity poses the requirement for systems to adapt to workload changes by provisioning and de-provisioning resources in an autonomic manner, such that at each point in time the available resources match the current demand as closely as possible [95].The benefits of an elastic architecture span low capital and operating costs and reliable Quality of Service (QoS) delivery [209]. Based on previous research on self-aware, autonomic systems [122, 47], an elastic system must first comprise monitoring capabilities to enable the collection of environment data and provide the functionality to track dynamic changes. Second, a proactive mechanism is important to anticipate the effect of changes and enable proactive planning of possible adaptation actions. However, as our findings in Section 3.2 show, predictions at design-time may not always be accurate.

Resource corrective actions can also happen in a *reactive* manner. With reactive schemes, allocation is adjusted on-demand based solely on recent run-time behaviour. At run-time, all system components are implemented and deployed in the production environment. This makes it possible to obtain much more accurate estimates of the various model parameters by taking into account the real execution environment [32]. Therefore the third component of the envisioned elastic architecture is a reactive mechanism, that adjust the resource allocations to align them with the actual environment conditions.

The efficiency of a reactive technique however, highly depends on its ability to respond to changes in a timely fashion. If the system cannot adjust quickly enough to environment changes, it will fail to perform adequately, making it hard to provide QoS guarantees in terms of performance and availability.

As Figure 3.9 suggest an elastic architecture must combine both predictive components to pro-actively plan for periodic environment changes as well as reactive components to refine design-time predictions and adapt the system to unprecedented volatilities. A monitoring unit is also necessary to track environmental informa-

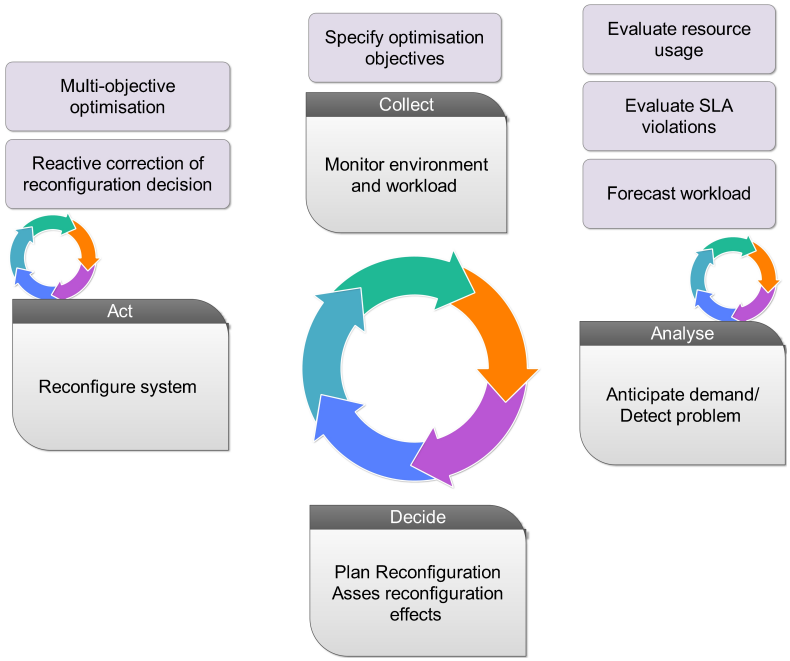


Figure 3.9: Elastic architecture vision based on [47].

tion and provide notifications for changes. This dissertation focuses on a reactive methodology to optimise the competing objectives of services performance and energy consumption after an environment change has been detected. The challenge is to estimate near-optimal configurations in a timely fashion, to refine possible previous proactive analysis and ensure that the allocated resources match the incoming workload at all times. In the following we convey our motivation to focus on this problem, through a use case scenario.

3.3 Use case Scenario

We consider the example of an Infrastructure Provider (IP) with distributed datacenters. Each datacenter is comprised of M heterogeneous PMs and N Amazon EC2 VMs of different capabilities and costs. A Service Provider (SP) rents part of the infrastructure to provide web-hosting services (e.g, on-line newspaper) to its clients. Web hosting services are considered 3-tier (i.e., web, application and database). Repli-

cas of each tier are hosted in each VM. The allocation of services to VMs and VMs to PMs forms a cloud *configuration*. Replicated tiers are balanced by load-balancers, also hosted in separate VMs. The workload of the services is represented by the mean request rates for each application tier. We further consider a SLA which sets the maximum service response time threshold at 120 seconds [3].

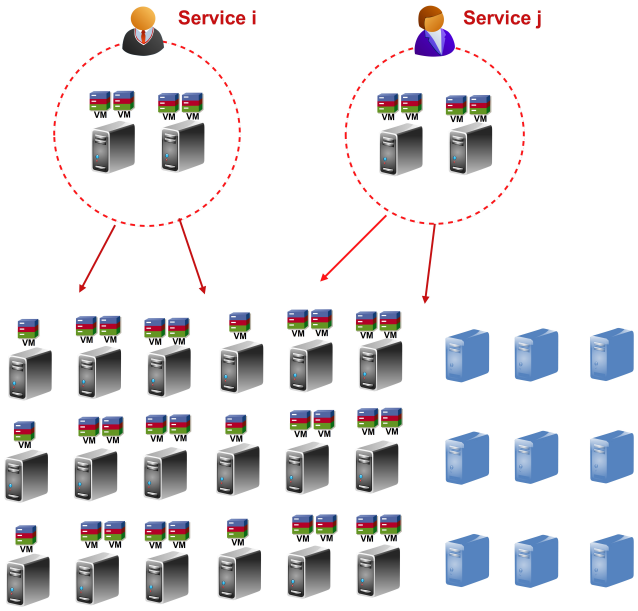


Figure 3.10: Cloud configuration example.

The best interest of SP is maximum services performance measured as minimum mean response time. Therefore, SP anticipates unlimited infrastructure scaling to support every possible incoming workload size. On the other hand, the more workload is being served, the more infrastructure resources must be activated to host it, increasing energy costs. While the IP must be able to provide sufficient infrastructure resources, his budget is constrained. Therefore the IP's best interest is to keep operating costs (such as energy consumption) controlled. Hence, the first challenge is to identify a near optimal configuration, to balance trade-offs between performance

[3]We base our selection of maximum response time SLA on the SLA used by Gambi in [77]. The selected threshold is similar, to real application SLAs. For example, Rackspase cloudfiles [98] provide a worse case response time of 180 seconds [18] while an average of 100 seconds has been observed for amazon aws services [128]

requirements and available budget. Figure 3.10 shows a highlevel configuration example, where part of the infrastructure serves the requested (servers indicated with gray) service instances while another part is deactivated to regulate energy consumption (servers indicated with blue).

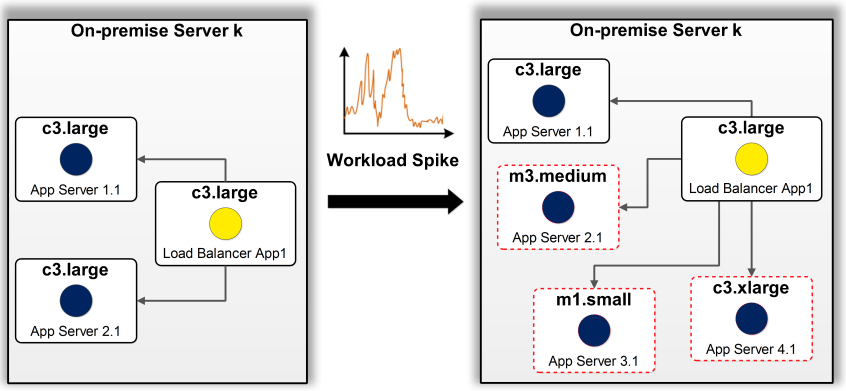


Figure 3.11: Reconfiguration to match increasing workload.

Changes in the environment will affect the initially deployed configuration. In case of a load surge, new servers must be activated to ensure compliance with the SLA while considering energy costs. Figure 3.11 shows a reconfiguration example; The PM server k initially hosts two EC2 c3.large VMs (2 vCPUs, 3.75 Memory GiB). To support the workload increase one initial VMs is resized to c3.xlarge (4 vCPUs, 7.5 Memory GiB) while a new m3.medium (1 vCPU, 3.75 Memory GiB) and m1.small (1 vCPU, 0.615 Memory GiB) are switched-on. It is critical that the new configuration will be decided and deployed within the SLA threshold of 120 seconds. Otherwise, IP risks revenue losses due to SLA violation penalties.

Driven by the law of supply and demand, consumer's requirements may also dynamically vary with time, which sets the requirement for dynamic SLA negotiations. Service requirements can change over time, due to e.g., continuing changes in business operations and operating environment, and thus may require amendments of original service requests. To support this, configuration management tools must output sets

of solutions in the space of the optimisation objectives to enable "what-if" analysis, rather than a single deterministic solution [191].

Reasoning about the pool of possible configurations for e.g., a cloud of 5 heterogeneous PMs and 10 VMs reveals the problem complexity. Considering up to e.g., 4 Amazon EC2 instances to configure the VMs there are 4^{10} (re)-configuration possibilities. If the system administrator further decides to switch on 2 more PMs and rebalance the VMs to the 7 now active PMs there are up to $7^{10} = 282475249$ additional reconfiguration options. Assuming that the evaluation of each configuration candidate takes 1 second (that is a low threshold as e.g. Koziolek [124] refers to $5-50$ seconds evaluation times), full design space exploration will take ~ 9 years. The design space of this trivial example is overwhelmingly large to be neither manually nor fully explored.

The goal of this thesis is to determine how to automatically adapt cloud infrastructure configurations to dynamic changes at runtime. Our focus in this problem is achieving a useful balance between the feasible optimality of architecture candidates and the timeliness of the optimisation procedure.

3.4 Problem Statement

Our goal is to produce near-optimal configurations that simultaneously honour the IP and SP non-functional requirements.

Given: A cloud datacenter is comprised of M heterogeneous PMs and N VMs of different capabilities, hosting the SP's multi-tier web services. A *workload profile* W describes the services request arrival patterns, resource consumption demands and runtime variability. A set of q quality objectives defines the stakeholder quality requirements $Q = \{Q_1, ..., Q_q\}$. Finally an SLA defines that the maximum services

response time threshold is T.

Problem: Find within time $t \leq T$ a set of configurations to host the web services workload W, that differ in their non-function trade-offs according to Q.

We consider the following two ($q = 2$) QoS metrics for optimising cloud configurations:

- The *mean response time* Q_{RT} which measures the mean time needed to serve a given web services workload. This QoS metric represents the SPs' requirements for increased availability and scalability to seamlessly manage workload surges. Amazon for example, has reported that 100 milliseconds of latency costs 1% drop in sales [85].

- The *energy consumption* Q_E of a configuration which measures the overall energy consumed by the physical and virtual servers in the cloud datacenter to serve a given workload. This QoS metric represents the IPs' requirement for controlled operating costs.

3.5 Problem Characteristics Summary

As showed in the description of our use case scenario in Section 3.3, deciding at runtime how to optimally modify cloud datacenter configurations is a challenging task. First, an optimal configuration must exhibit a good trade-off between stakeholder non-functional goals which often mutually conflict with each other. For example as Figure 3.12 shows, more hardware servers can accommodate higher loads and therefore improve the services performance (measured e.g., in Service Level Agreement (SLA) violations) but also cause increased energy costs and vice versa.

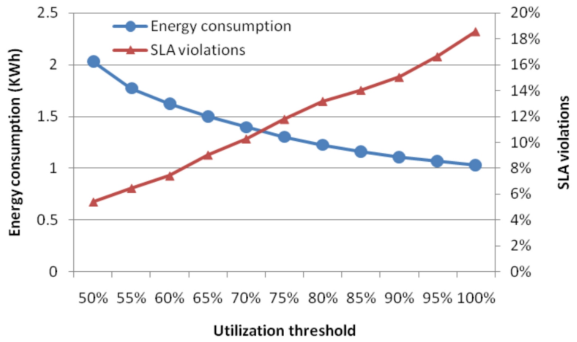


Figure 3.12: Energy-performance trade-off in a cloud of 100 PMs serving Web services' load [33].

Additionally, we seek to provide an automated decision support tool that will help stakeholders take fast and well-informed design decisions by showing multiple near-optimal trade-off solutions in the space of the optimisation objectives.

Second, the space of all possible datacenter configurations is huge due to the number of the available configuration parameters. Each machine contains a large number of configurable parameters, such as CPU time, memory, and network bandwidth, and the VMs on the same host may interfere with each other. Therefore, using a manual or brute force search for finding an optimal configuration is infeasible. In practice, only a limited set of configurations can be evaluated. Third, clouds are highly volatile; parameters as workload demand, resource availability and energy price might change at run-time. Such changes affect the underlying optimisation problem instance and therefore the initially chosen datacenter configuration must be updated. Finally, the cost of the optimisation process itself makes runtime configuration adaptation complex because of the additional time and computational resources needed to find the improved cloud configurations. Often it may be better to make a suboptimal decision quickly rather than invest time and energy searching for savings that are not enough to recoup the investment [114]. Overall, it is critical that the optimisation process will not consume more resources than it can save.

3.6 Use-case Simulation Settings

The target system is an IaaS enterprise cloud environment, expected to create a view of infinite computing resources to users by optimising the trade-offs between Q_{RT} and Q_E. It is therefore important to evaluate the proposed methodology on a large-scale enterprise datacenter infrastructure. However, it is extremely difficult to conduct repeatable large-scale experiments on a real infrastructure, which is required to evaluate and compare the proposed methodology. Therefore, to ensure the repeatability of experiments, simulations have been chosen as a way to evaluate the performance of the proposed heuristics.

In this work, we have used CloudSim [36] to simulate cloud datacenters as it is the single currently mature cloud simulation framework. In contrast to older alternative simulation tool-kits such as GridSim [34], and GangSim [60], CloudSim allows the modelling of virtualised environments, supporting on demand resource provisioning, and management. Another advantage is the realistic modelling of energy consumption, using real data providedby the SPECpower benchmark [4]. As Figure 3.13 illustrates, CloudSim features a multi-layer design for modeling and simulation of virtualized cloud-based datacenters. The CloudSim simulation layer provides dedicated services, VM and PM management interfaces as well as memory, storage and bandwidth control policies. Fundamental issues as provisioning of hosts to VMs, scheduling resources among co-hosted VMs, managing the services execution and monitoring the dynamic cloud datacenter state are also handled by this layer. The up-most layer in the CloudSim stack enables the generation of workload request distributions (also referred to as cloudlets) and the specification of customised cloud configurations.

We consider typical datacenter computational capabilities [189, 22]; for PMs we simulate HP ProLiant ML110 G4 (2 cores × 1860 MHz), HP ProLiant ML110 G5 (2 cores × 2660 MHz), IBM x3470 (4 cores × 2933 MHz), IBM x3250 (4 cores × 3067

[4] `http://www.spec.org/power_ssj2008/`

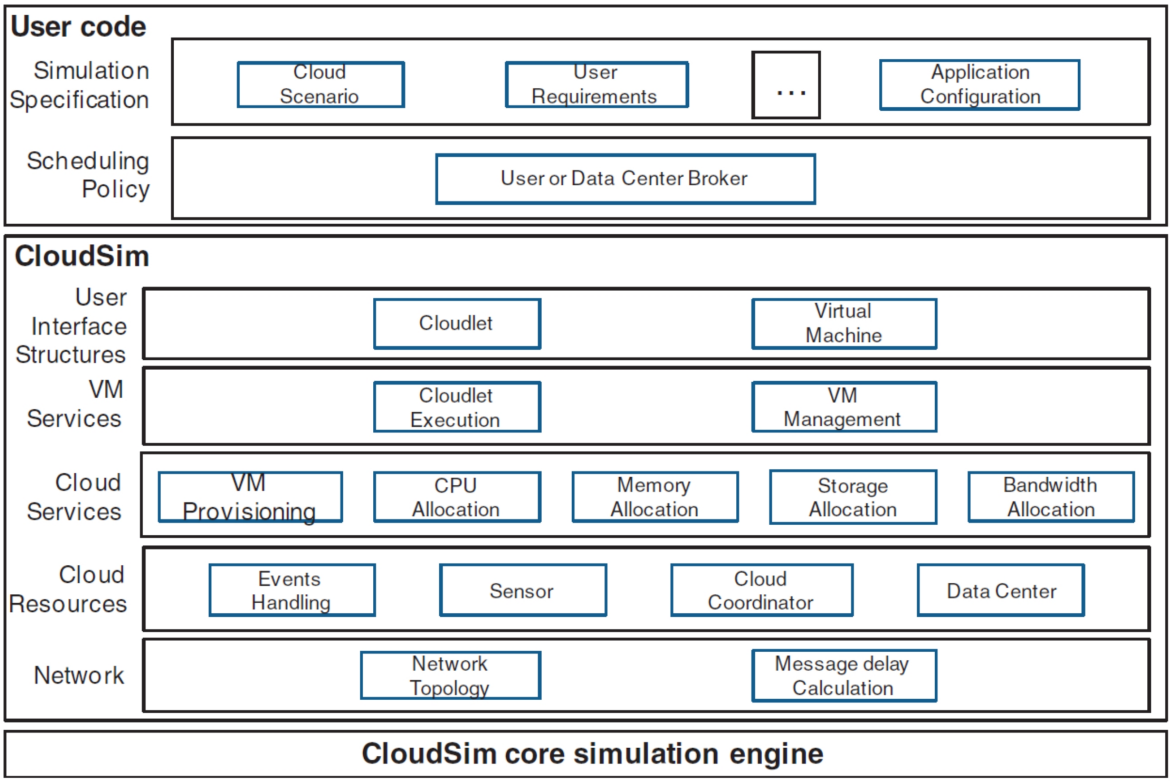


Figure 3.13: The CloudSim layered architecture [36].

MHz), IBM x5670 (6 cores × 2933 MHz) and IBM x5675 (6 cores × 3067 MHz). Using CloudSim simulation, the frequency of the servers' CPU are mapped onto (Million Instructions Per Second) MIPS ratings [22]; 1860 MIPS each core of the HP ProLiant ML110 G4 server, 2660 each core of the HP ProLiant ML110 G5 server, 2933 MIPS each core of the IBM x3470 and x5670 servers , 3067 MIPS each core of the IBM x3250 and x5675 servers. Each server is also modeled to have 1 GB/second network bandwidth.

The VM characteristics correspond to the following Amazon EC2 instances [5]: high memory extra large, high CPU medium, extra-large, small and micro. The difference is though that the modelled VMs are single-core. The CPU and RAM characteristics for each VM type are: high-CPU medium instance (1000 MIPS, 1.7 GB), extra large instance (2000 MIPS, 3.75 GB), small instance (1000 MIPS, 1.7 GB)

[5] `http://aws.amazon.com/ec2/instance-types/`

and micro instance (500 MIPS, 613 MB) [22].In our experiments we consider cloud sizes combining $100-1000$ PMs and $100-1000$ VMs, representing commercial cloud settings.

Recent findings [40, 166] from Google trace analysis provide a broad characterization of cloud workloads, reporting heavy-tailed job durations and job shapes. Therefore, we model workloads as synthetic Bag-of-Task (BoT) [105]. According to BoT cloud services are considered black-box resource requests with Weibull arrival patterns, sizes and runtimes [49]. Figure 3.14 shows a BoT workload example, comprised of four BoTs.

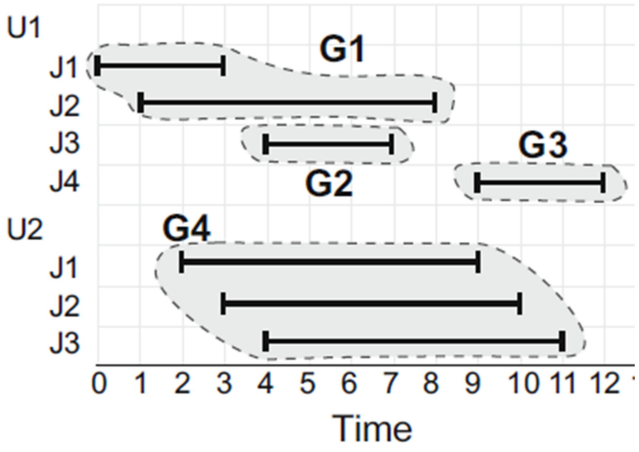


Figure 3.14: BoT example. The Y axis shows different users and the X axis shows the time. We observe that user U_1 has submitted three BoTs; G_1, G_2 and G_3 while the user U_2 has submitted a single BoT G_4. The BoTs G_2 and G_3 contain one task while the BoT G_1 contains two tasks and the BoT G_3 contains three. The individual tasks within BoTs may have different staring times as well as different runtimes [105].

The BoT workload model is decoupled from the actual services architecture and components since only request resource sizes, arrivals and runtimes are required. Hence it is a realistic model for IPs because due to legal boundaries internal services information is often restricted from infrastructure providers [149]. In this study we examine the effect of workload sizes with $[20000-90000]$ tasks on the cloud configurations. This workload triggers scaling up and down scenarios for our cloud. In Section

7.3 we discuss in more detail the implementation we followed for the generation of BoT workloads.

In our simulation environment, the BoT tasks are assigned to the available VMs in the datacenter for execution. The VMs with lowest CPU utilisation are prioritised for hosting tasks. Each VM may host more than one BoT tasks. A BoT task models a cloud service and is essentially a request for CPU and memory resources with a corresponding runtime, that defines the total execution time of the particular service. We consider that each BoT, possibly comprising many tasks, is a composition of services providing a higher-level functionality to the end-user. After all tasks have been mapped to the VMs, the CloudSim VM scheduler is responsible for allocating the requested resources to meet the demand. The BoT tasks compete for the available resources of the VMs at each time. In case there are not sufficient resources available to enable execution, the VM Scheduler places the tasks in a queue. Each VM is associated with its own queue depending on the tasks assigned to it. Periodically, the VM scheduler checks the resources availability and changes the status of tasks from queued to executing.

3.7 Summary

Cloud computing has evolved from a series of mature technologies, that include virtualisation, autonomic, grid and utility computing. By implementing elasticity cloud systems can closely adapt to environment changes by provisioning and de-provisioning their resources accordingly. As a result higher server utilisation rates can be achieved, which alleviates the problem of over-provisioned infrastructure and reduces capital and operating costs of organisations. There are two general approaches to dynamic resource provisioning: proactive and reactive. The first allocates resources in advance of environment changes and as a result the system can seamlessly adapt. Proactive

techniques provide very good performance when the changes are periodic. However they fail when predictions are not possible due to e.g., unforeseen patterns or noisy data. In reactive allocation the resources are updated at run-time after a workload change has been detected. Reactive provisioning is attractive as the resource allocation is based on recent system behaviour and therefore can enable highly accurate estimates. However the efficiency of reactive schemes practically depends on their ability to respond to changes in a *timely fashion*.

Both proactive and reactive components are crucial in an elastic architecture implementation to both pro-actively plan changes and fine-tune the predictions at run-time to improve performance and resources availability. In this dissertation we focus on a reactive framework to optimise the quality objectives of services mean response time (Q_{RT}) and cloud energy consumption Q_E in a cloud datacenter. The problem challenges may be summarised as:

- large design space exploration
- environmental volatility
- trade-off optimisation
- convergence of the optimisation process at run-time

Chapter 4

Formalisation of the Cloud Design Space

THIS thesis describes a framework for optimising at runtime the energy consumption (Q_E) and services response times (Q_{RT}) of IaaS clouds, running mutli-tier web applications. Manually checking for possible configuration alternatives in a trial-and-error approach for improvement, would be laborious and error-prone. Additionally human intervention may result to overlooking valuable solutions because of bias [88]. To overcome these limitations, we discuss in this chapter how our framework can benefit from an automated process of improving configurations. We apply Model Driven Engineering (MDE) concepts to formalise the cloud configurations domain and specify which changes to configurations are valid, towards tuning their non functional qualities without interfering with their functionality. The work presented in this chapter has been published in [43].

4.1 Towards an Automated Configurations' Improvement Method

Following Fleurey et al.'s [1] definition,

> " a model is a collection of objects and relationships between them that together provide a representation of a real system."

MDE proposes a move away from human interpretation of high-level models such as sketches and natural language, towards a more automated process where structured models are used as the first artefacts of the software engineering process [1]. While unstructured models are useful for communication they cannot be used with automated processes due to their ill-defined semantics. Instead, in MDE a model's structure is defined by a *meta-model*, that is another model to specify the concepts and constraints available to the models. The use of models in complex software engineering problems abstracts away from the implementation complexity and enables a stronger understanding of the problem domain. Additionally developing a model enables reasoning about a system without suffering the cost of actually implementing it [142], hence higher productivity can be achieved. The process is also maintainable because the domain can be easily updated as requirements change.

The goal of an automated improvement process is to find meaningful configuration alternatives in the design space. We define as the *design space* the set of all configuration models reachable by the automated improvement method [124]. However, to improve an input cloud configuration model, the automated method requires a formulation of how this model may *change* in order to find improved alternatives in the design space. Configurations cannot change arbitrarily but must adhere to predefined design constraints, to ensure that they are viable. Additionally after changes, the configuration functionality must remain unmodified and the system must still be realisable[124].

To enable an automated method for cloud configurations optimisation, we propose a metamodel to describe the cloud domain semantics, focusing on the IaaS layer. We also define how the configuration design can be changed to tune the quality without affecting the functionality by formalising the concept of *Degrees of Freedom (DoF)*.

The proposed metamodel aims to formalise the achievable cloud design space. This enables subsequently, the use of complex optimisation methods such as metaheuristics, to efficiently explore all reachable configuration models by tuning the formalised DoFs.

The Object Management Group[1] (OMG) have defined a standard for a metamodelling architecture, called the Meta-Object Facility (MOF) [156]. The architecture is composed of four layers: The top layer, $M3$, provides a metamodelling language for specifying metamodels in the second layer,$M2$. The Unified Modeling Language (UML) [2], the de facto modelling language, is an example of an $M2$ metamodel. The third layer, $M1$, contains models that conform to metamodels in $M2$, for example UML class diagrams. Finally $M0$ is the object layer - i.e. the real-world problem being modelled [199].

To design the cloud metamodel we have used the *Eclipse Modeling Framework*. The Eclipse Foundation [2] have implemented their own metamodelling architecture that aligns with MOFs four-layer architecture. The Eclipse Modeling Framework [3] (EMF) has a MOF-equivalent metamodelling language called *Ecore* and provides stable and well maintained tool support for modelling activities, such as a graphical editor for defining metamodels and tools for automatically generating model editors from a metamodel.

4.2 The Cloud Metamodel

In this section we describe the proposed cloud metamodel, that was designed utilising EMF. This metamodel has abstracted away from other cloud layers such as PaaS and SaaS. The focus here is on IaaS, that offers on-demand reconfigurable physical and

[1] `www.omg.org`
[2] `www.eclipse.org`
[3] `www.eclipse.org/emf`

virtual hardware resources as a service to stakeholders towards meeting fluctuating workload needs, allowing also for an optimum resource utilisation [188].

In current literature there have been different approaches to model resources according to the environment they are contained. For example PCM metamodel [20] provides a coarse-grained resource container, supporting component based software context. On the other hand, DMM meta-model [100] is a fine-grained description of virtualised datacenters. However, a specification of resources in the cloud domain has not been yet covered.

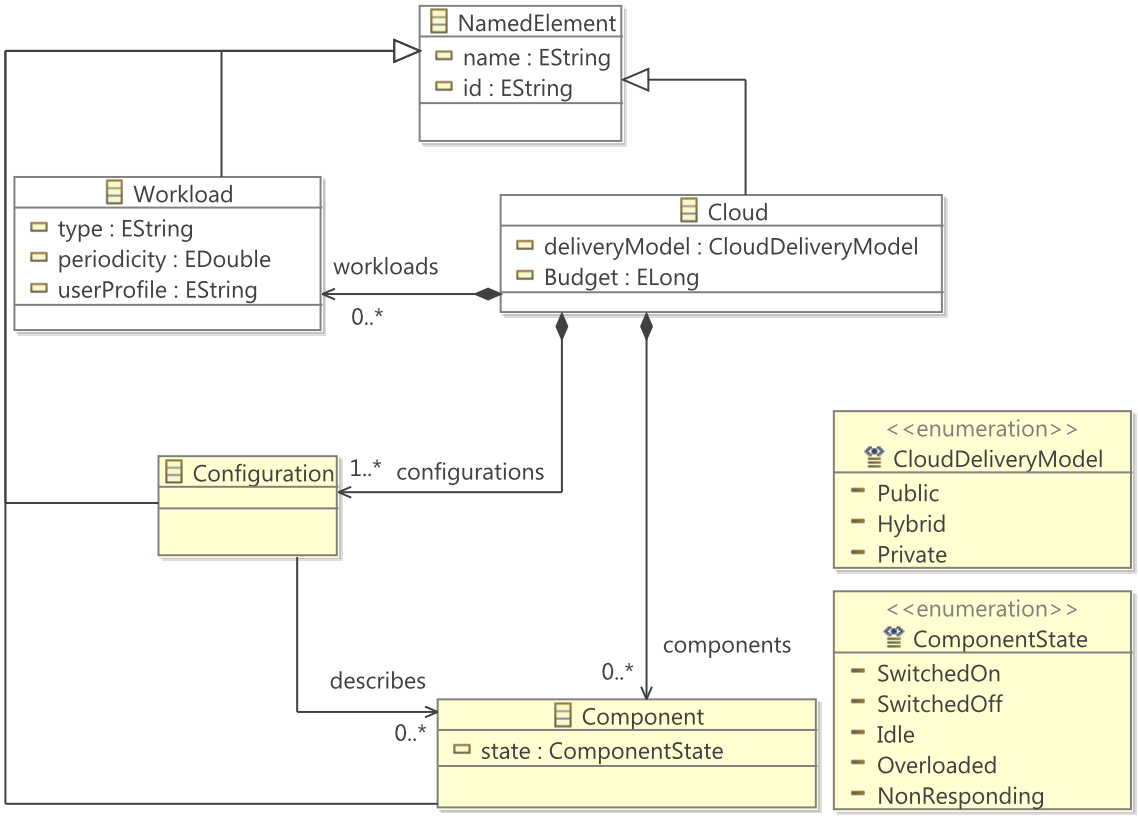


Figure 4.1: Cloud System Metamodel. Each box is a meta-class and the lines represent references between meta-classes. References and attributes of a meta-class are known as meta-features. The black diamonds on the relations represent composition, meaning e.g. in this Figure that Wrokload, Configuration and Component can only exist if they are contained in the Cloud. The asterisks on a reference specifies the multiplicity of this reference. Fianlly the white triangular arrowhead represents inheritance. In this exmple we see that both Workload and Cloud inherit from NamedElement.

The root entity comprising all the other entities is the *Cloud.* The *deliveryModel* property refers to the commercial Cloud realization (i.e., public, hybrid or private). More specifically a private cloud is provisioned for exclusive use by a single organization, a public cloud is provisioned for open use by the general public and a hybrid cloud infrastructure is a composition of two or more distinct cloud infrastructures (i.e., private and public) [147] The property *budget* denotes the provider's invested capital (e.g. in $). The budget will dictate how much the cloud can afford to scale in the face of unexpected workloads. A Cloud's purpose is to serve an arbitrary amount of *Workloads*. The *Workload* properties *type*, *periodicity* and *userProfile* specify the intensity and characteristics of the imposed workloads.

The Cloud can have many possible *Configurations*, denoting the system's hardware and software components as well as the possible mappings between them. The Cloud *Components* can be encountered in different states, as enumerated in *ComponentState*, indicating their health and availability. We distinguish five different states: "Switched-on" indicating that the component is in use; "Switched-off" indicating that the component has been deactivated; "Idle" indicating that no function or service is running on this component; "Overloaded" indicating that there are not enough resources in this component to complete a requested task; and "Non-responding" indicating a possible failure.

4.2.1 Cloud Components

The central entity *Component* abstracts the Cloud infrastructure. As Figure 4.2 shows, we distinguish between two component types, the *HardwareComponent* denoting the Hardware Infrastructure and the *SoftwareComponent* denoting the Software Infrastructure.

The software landscape can be summarized as *DeployedServices* executing on Virtual Machines (*VM*) while assisted by *RuntimeEnvironment* Entities, e.g., the

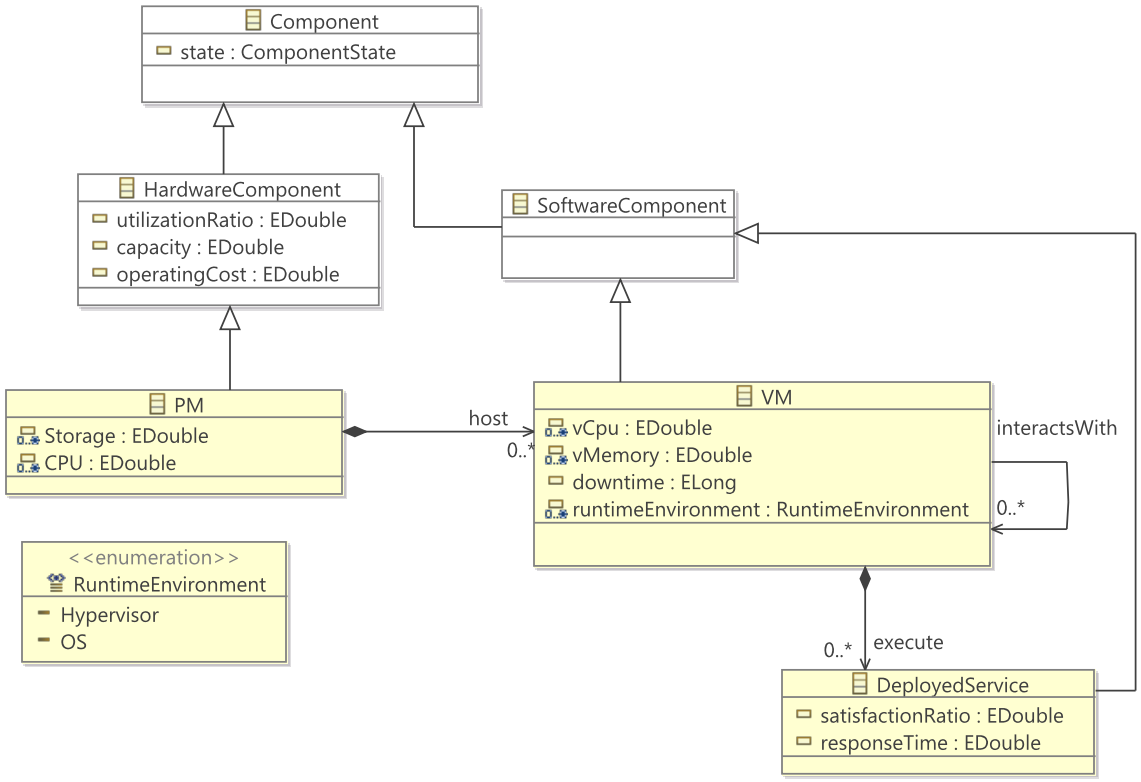


Figure 4.2: Cloud Infrastructure.

hypervisor. Other properties that the *VM* class entail, are *vCPU* i.e., the quantity of physical CPU assigned to it, *vMemory* i.e., the quantity of physical Memory assigned to it, and the *downtime* i.e., time period when a VM is unavailable as a VM migration is being carried out. The relation *interactsWith* models shared communication among VMs.

The *HardwareComponent's* properties *utilizationRatio capacity* and *operating-Cost* provide feedback relevant to the efficiency and cost of a configuration. Core entity of the hardware landscape are the Physical Machines (*PM*), comprised of *storage* and *CPU*. PMs can host multiple VMs during their operation.

4.2.2 Cloud Variability

In Figure 4.3 the volatility of the Cloud's context and its capability to adapt accordingly are depicted. Imagine for example, the case of two VMs sharing resources and assume that the workload of one increases leading to an SLA violation. The Cloud configuration will have to be *changed* by e.g., increasing the CPU size of the overloaded VM so as to support its hosted services without triggering SLA violations. In this section we specify the *degrees of freedom (DoFs)* of the configurations, i.e., the configuration parameters that are free to be varied to affect QoS, leaving the functional part unaffected. As the DoFs conform to the metamodel, we ensure that changes to configuration models will also comform to the metamodel and will be meaningful.

Cloud stakeholders may have many possible generic *High Level Goals* to guarantee the quality of the system as well as enforce economic considerations. Examples of quality attributes may be based on [124]:

- **Performance:** Performance is concerned with the timing behaviour and resource efficiency of the system. Important performance measures are response time of system services, resource utilization, and throughput.

- **Reliability:** Reliability is the capability of a system to provide functionality as expected for a specified period of time in the intended execution context. It is for example measured as the probability of failure on demand.

- **Modifiability:** Modifiability is concerned with the costs of changing the system, e.g. if new functionality should be added or if corrective changes are made.

- **Security:** Security is the capability of the system to resit unauthorized usage, i.e. to protect sensitive data and services so that only authorized users can access them.

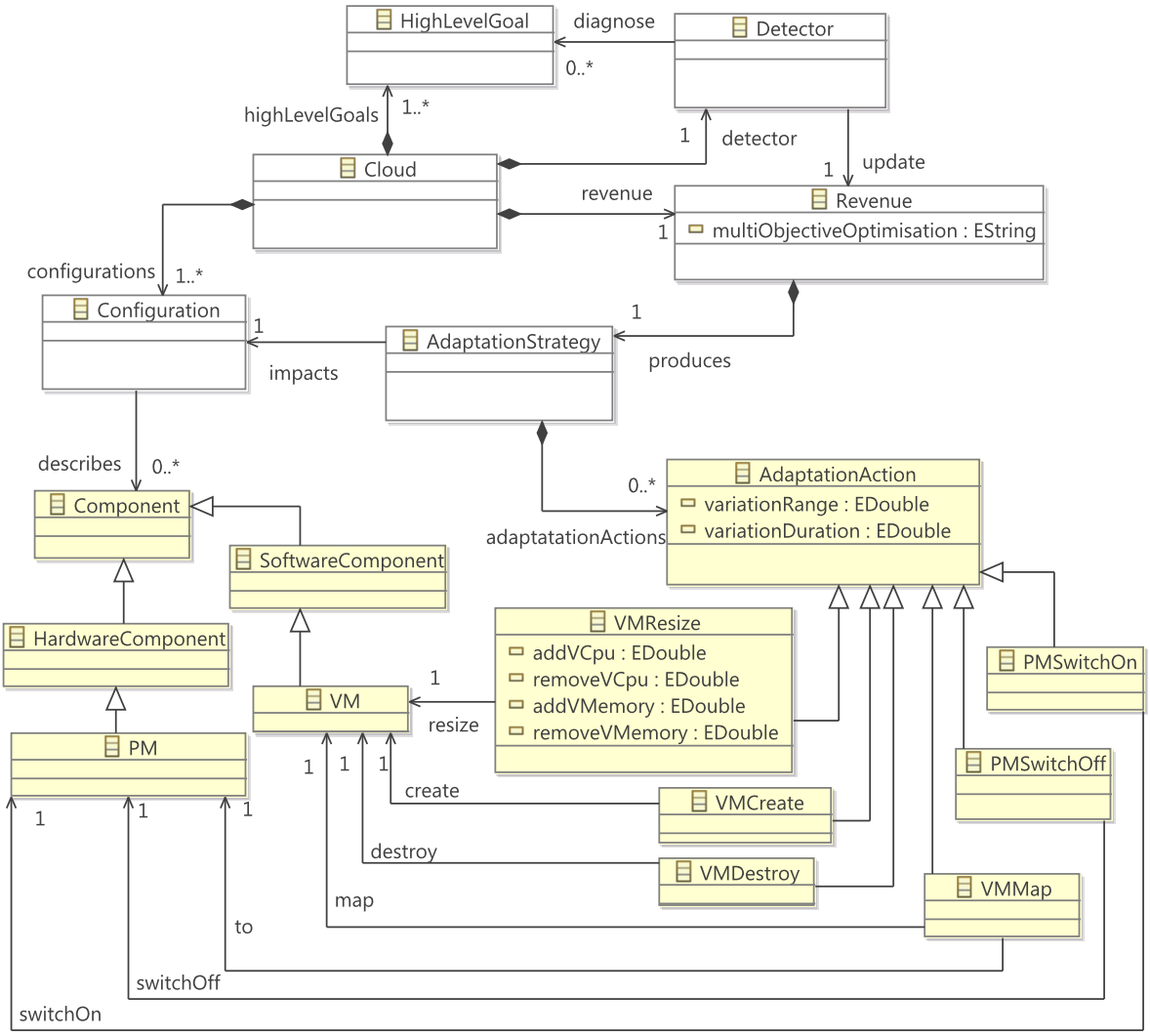


Figure 4.3: Cloud Variability.

- **Testability:** Testability describes how well the software can be tested to detect faults.

- **Usability:** Usability describes how easy users can work with the system and accomplish their tasks

Additionally,the economic considerations may comprise [124]:

- **Costs:** Costs are the main quality to trade-off against the software quality attributes named above. What types of costs need to be considered depends

on the organizational context: Usually, the direct development costs have to be considered. Additional costs are maintenance costs, hardware procurement costs, operating costs, or licensing costs.

- **Monetary Benefit:**The benefit to be achieved by the developed software system can be quantified and compared to the expected costs, to calculate the return-of-investment

- **Time to Market:**Development time may be important if a new type of system is developed that is supposed to capture a share of an emerging new market.

Many of the aforementioned quality attributes are in mutual conflict with each other: For example, security and reliability often negatively influence each other: While a system is secure if it offers few places that keep sensitive data, such an organization may lead to single points of failure and decreased reliability.

To specify a meaningful optimisation function of reasonable complexity, the *Detector* interface diagnoses the most relevant subset of *High Level Goals* with respect to the current stakeholder requirements and forms a corresponding multi-objective optimisation problem, modelled by the *Revenue* entity. The Revenue denotes a - closely aligned with the runtime context - objective function, expressing the quantity or quantities that have to be currently maximised (e.g., server utilisation ratio) or minimised (e.g., network Traffic) to achieve the required system configuration. A straightforward way to express an objective function in real-world systems where an exact formula to correlate system properties with quality does not exist [112] is the "Metrics as Fitness Functions" approach (MAFF) [86]. According to MAFF, metrics are regarded as means towards evaluating a property of interest, allowing metrics to both assess and improve a system. More details are discussed in Section 5.1.4.

The realisation of the Revenue yields an *AdaptationStrategy*, which provides means towards a Cloud reconfiguration. The *AdaptationStrategy* is implemented

via a concatenation of *AdaptationActions*. The *AdaptationActions* entity captures the DoFs of a Cloud resources landscape describing the types of actions that can be performed towards transitioning from an old configuration to an improved candidate. The identified possible action types span: (a) *VMResize* i.e., how much virtual memory or virtual CPU should a VM receive (b) *VMCreate* i.e., creation of a new VM (c) *VMMap*: (re)selection of PMs to host the aforementioned VMs (d) *VMDestroy* i.e., switch off a VM (e) *PMSwitchOn* i.e., activation of a new PM (f) *PMSwitchOff* i.e., switching-off a PM.

The property *variationRange* specifies the range in which the aforementioned entities may vary, while the property *variationDuretion* indicates time limits for the reconfiguration actions.

Figure 4.4 shows a simple cloud configuration example. The configuration is comprised of three PMs ($PM0$, $PM1$, $PM2$) hosting two intercommunicating VMs ($VM0$, $VM1$), each one executing a different enterprise service ($DeployedService0$, $DeployedService1$). Let us assume an energy optimisation problem, declared in the *Revenue* entity. The *Adaptation Strategy* contains a series of DoF changes to tune the quality of the configuration towards meeting the *Revenue* objective. In 4.4, the reconfiguration actions to tune the energy targets of the current model include: *Switch-off* $PM1$ which does not currently host any VMs, and *migration* of $VM0$ from $PM2$ to $PM0$.

4.3 Summary

In this chapter, we discuss how stakeholders can be supported by an automated cloud configuration improvement method, that enables them to reason about QoS attributes and map them to the configuration design. A leading question is how configuration models may be changes automatically. The main requirements for the

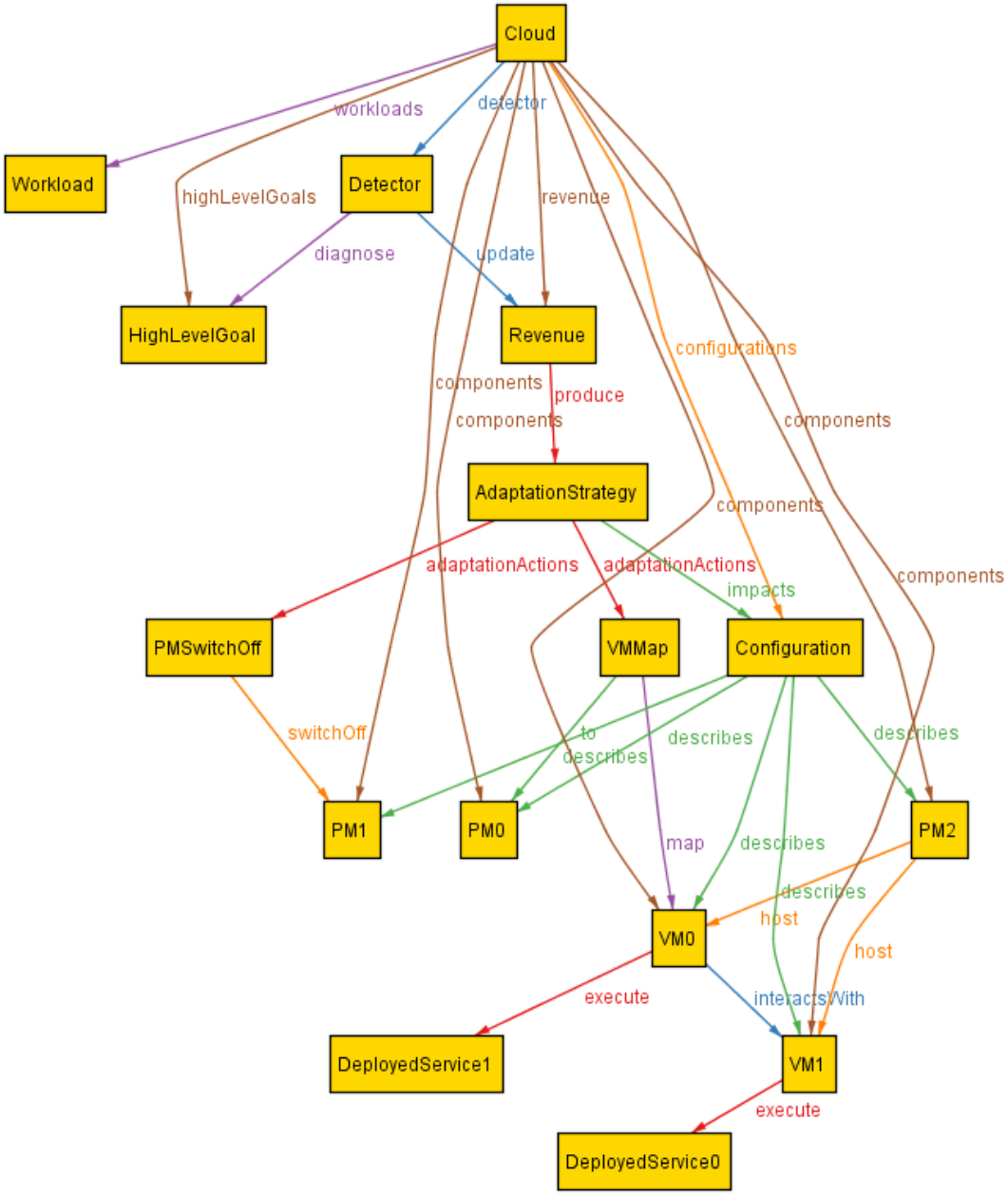


Figure 4.4: A configuration model instance example.

automated variation of cloud configurations is that (i) the changes must capture influential factors on quality properties and (ii) after the changes are applied to the initial model, the result must be a meaningful alternative with unchanged functional behaviour. To this end, we have introduced the concept of degrees of freedom (DoFs), that describe independent ways a given configuration model may be varied.

We have presented our metamodel, designed with EMF, to describe the cloud configurations structure with focus on the IaaS perspective. Our metamodel also specifies the DoFs of configurations. By tuning candidates along the DoFs, complex heuristics may be used, as Chapter 5 discusses next, to efficiently explore the problem design space, towards improving the QoS metrics of interest.

As will be discuss in the next Chapters, this thesis has focus on the runtime optimisation of the response time (Q_{RT})and energy consumption (Q_E) metrics. However the proposed approach may be generalised to different quality objectives, that may be specified in the the ***Detector*** entity of the cloud metamodel.

Chapter 5

Search-Based Optimisation of Cloud Configurations

Search-Based Software Engineering (SBSE) [91, 89, 88] is an approach that treats software engineering problems as optimisation problems. Software engineers often face problems associated with the balancing of competing constraints and managing conflicting trade-offs between requirements. Perfect solutions are often either impossible or impractical and the nature of the problems often makes the application of analytical algorithms problematic. SBSE techniques can help discover acceptable solutions to these problems, based on the observation that while it might be infeasible to identify a precise "best" solution to the problem, it is still possible to decide which solution is "better" among a set of candidate solutions [89].

The term was coined in 2001 by Harman and Jones [89] and has ever since received exponential popularity in the literature. Numerous publications show that SBSE has now permeated almost every area of software engineering activity such as software testing [90], software architecture optimisation [6], requirements engineering [12] and project planning [68].

After an overview of the key concepts of SBSE in Chapter 2.1.2, we focus in this Chapter on the key areas of relevance of SBSE for this thesis: representing and re-formulating the cloud configuration optimisation problem as a search problem. The proposed SBSE reformulation of the cloud configuration optimisation problem, detailed in Section 5.1, has been published in [44].

5.1 Reformulating Cloud Configuration Optimisation as SBSE Problem

We have chosen to utilise a linear, integer-based genotype, structured in a way that allows the complete expression of cloud configuration models, as specified in Chapter 4. A linear genotype enables coupling and cohesion of the domain class diagrams, as each gene represents a different feature in the class diagram. A linear representation has the advantage that the plethora of existing linear genotype-based search algorithms, such as genetic algorithms, evolutionary strategies, simulated annealing and hill climbing can be readily used [143] without the need to define more complicated genetic operators or new metaheuristic techniques[199].

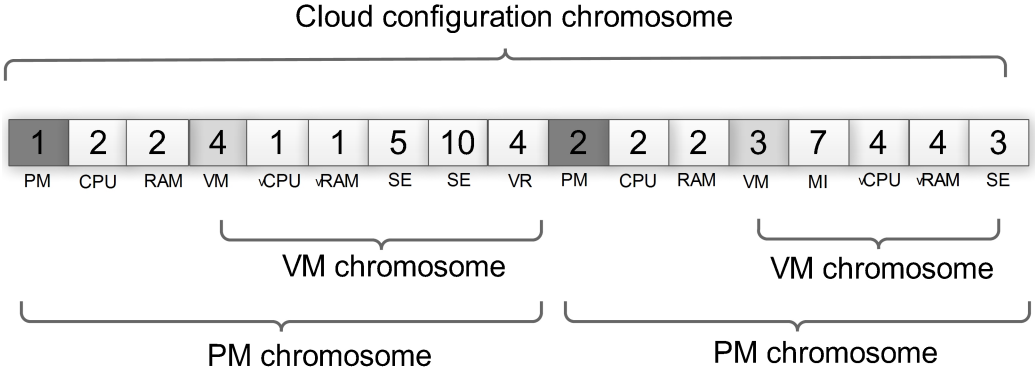


Figure 5.1: The genotype of our representation

Figure 5.1 shows the genotype that we have defined to encode cloud configurations. The genotype is comprised of a number of chromosomes, that represent the

configurations' constituents. Three chromosomes have been specified to describe a configuration: *Cloud Configuration (CC)*, the *Physical Machine (PM)* and the *Virtual Machine (VM)*. The CC chromosome defines a container for the PM and VM chromosomes to be grouped together, to express the architecture of configuration solutions. Different cloud architectures may comprise different computational capabilities resulting from variable VMs and PMs combinations. As a result genotypes may also exhibit variable lengths to express corresponding heterogeneous cloud configurations. The length limits are configurable and depend on user preferences for the number of available computing nodes. As described in Chapter 3.6, we consider in this work, configurations comprising 100-1000 VMs and 100-1000 PMs. Therefore genotypes' lengths may comprise a minimum of 100 PM and VM chromosomes and a maximum of 1000 PM and VM chromosomes.

ID		**Description**
PA	PM Activate	Hardware has an effect on quality attributes. More active servers can make the datacenter perform better.
PD	PM Deactivate	Deactivation of low utilised servers provide a way to negotiate trade-offs such as datacenter speed, reliability, cost.
VR	VM Resize	The capabilities of a VM can be resized. For example a VM may receive a bigger CPU slice from its host to enforce its processing power.
VC	VM Create	New VMs can be instantiated to exploit more processing capacity and increase parallelism.
VD	VM Destroy	VMs may be destroyed to release its reserved resources to promote cost saving policies.
MI	VM Migration	VM migration can alleviate resource contention issues from overloaded servers or redistribute the VMs to release idle resources.

Table 5.1: Formalised Degrees of Freedom (DoF) in Cloud Resource Configurations.

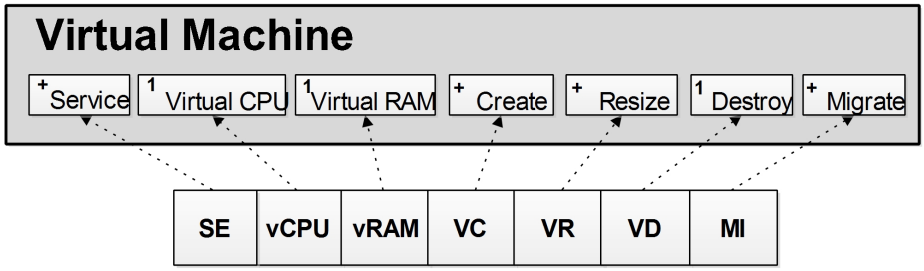
We have already mentioned in Section 2.1.2 that the building blocks of chromosomes are specified by genes. Here, the genes are used to model two different dimensions of cloud configurations: (i) the *building blocks* of VMs and PMs, as presented in Chapter 4, such as e.g. CPU and virtual CPU and the (ii) *Degrees of Freedom*

Gene	Unit	Description	Chromosome
CC	$\mathbb{N}$	Cloud configuration id	CC
VM	$\mathbb{N}$	Virtual machine	PM
SE	$\mathbb{N}$	Service id	VM
vCPU	MIPS	Virtual CPU id	VM
vRAM	MB	Virtual RAM id	VM
VR	$\mathbb{N}$	VM resize DoF	VM
VC	$\mathbb{N}$	VM create DoF	VM
VD	$\mathbb{N}$	VM destroy DoF	VM
MI	$\mathbb{N}$	VM migrate DoF	VM
PM	$\mathbb{N}$	Physical machine entity	CC
CPU	MIPS	Physical CPU capacity type	PM
RAM	MB	Physical RAM capacity type	PM
PA	$\mathbb{N}$	PM activate DoF	PM
PD	$\mathbb{N}$	PM deactivate DoF	PM

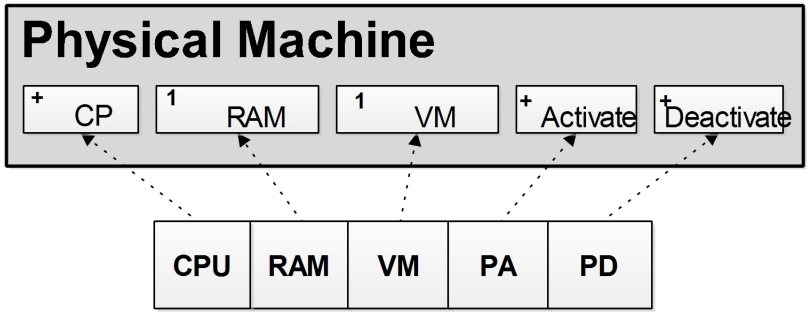
Table 5.2: Design of used genes in the chosen representation.

of the configurations (DoFs) i.e., which adaptation actions may be applied to a configuration towards tuning its quality without affecting the functionality. Varying the specified DoFs during the search achieves reconfigurations of the cloud genotypes. As the DoFs define the reconfiguration points of the solution candidates, they are also the only parts of the genotype that may change during the metaheuristic search. All specified genes used to form our configurations encodings are summarised in Table 5.2. The genes correspond to meta-features and chromosomes correspond to meta-classes of our proposed cloud meta-model [43] discussed in Chapter 4, for specifying the structure of IaaS configurations. Conformance to the meta-model structural rules ensures that genetic operators applied to configurations will always lead to meaningful solution alternatives, enabling the engineering of an automated method for design space exploration.

The figures 5.2a and 5.2b illustrate the chromosomes for encoding a VM and a PM node. Dark coloured boxes indicate chromosomes while light indicate genes. A VM may comprise one or more service genes, a vCPU and vRAM which correspond to a slice of the physical CPU and RAM that the VM is allowed to access. As discussed in



(a) The VM chromosome.



(b) The PM chromosome.

Figure 5.2: Overview of VM and PM chromosomes. Symbols 1 and + indicate that the elements occur exactly once and more than once respectively.

Section 3.6 our services follow the BoT model, and therefore service genes correspond to black-box resource requests for vCPU (in MIPS) and vRAM (in GB). A VM may be reconfigured via the *Resize (VR)*, *Create (VC)*, *Destroy (VD)* and *Migrate (MI)* genes. These genes encode the relevant as DoFs described in Table 5.1. As described in Section 3.6 the simulated VM in our study may exploit the following CPU and RAM configuration settings; (1000 MIPS CPU, 1.7 GB RAM), (2000 MIPS CPU, 3.75 GB RAM), (1000 MIPS CPU, 1.7 GB RAM) and (500 MIPS CPU, 613 MB RAM). Hence, the regarding the *Resize (VR)* reconfiguration DoF, our framework supports three different CPU and RAM resizing units; The possible CPU resize units are 2000, 1000, and 500 MIPS while the possible RAM resize units are 3.75 GB, 1.7 GB and 613 MB. The resize actions are decided at random, and applied only if the host PM has enough available resources to support the new VM configuration.

A PM comprises one or more CPU cores, RAM capabilities and may host one or more VMs. A PM is reconfigured via the *Activate (PA)* and *Deactivate (PD)* genes, which correspond to DoFs in Table 5.1 accordingly.

Coming back to the genotype example in Figure 5.1, we observe that this cloud configuration encoding contains two PMs with ids 1 and 2, each hosting a single VM. The services with ids 5 and 10 run on the VM with id 4 while the service with id 3 runs on VM with id 3. As discussed in Section 3.6, a service is modelled as a resources request (e.g., 1000 MIPS). The VM 4 is assigned a resize reconfiguration rule, and the VM 3 is assigned a migration reconfiguration rule to the PM with id 7.

5.1.1 Initialisation

In the general case, the initial search population of a population based algorithm such as a genetic algorithm is generated by assigning random values to each gene. This would spread the initial population randomly about the search space, allowing a wide area to be explored [199]. The selection of the initial population can be also done in more constructive ways. For example, depending on the problem it may be beneficial to seed the population with variants of an existing model or with a-priori known possible solutions. It has been recognized that if the initial population to the genetic algorithm is good, then the algorithm has a better chance of converging to an optimal or near optimal solution. To the contrary, if the initial supply of individuals is not large enough or not good enough, then it might be difficult for the algorithm to explore high quality solutions [161]. Regarding the size of the population, if it is too small the GA may not adequately explore the search space while if it is too large, the algorithm may take a long time to converge [136].

To balance the trade-off we pick an initial population of 100 . This is a commonly used population size setting [160, 185] and consistent with several literature studies that suggest the use of a population size of 50 for each optimisation objective [136].

In our case, input to the GA is a cloud configuration c_i to be optimised with reference to the non-functional requirements Q_{RT} and Q_E, that as discussed in Section 3.4 stand for response time and energy consumption respectively. To inject a portion of possibly improved quality solutions in comparison with the starting configuration input, random mutations are introduced to c_i. The mutations are achieved by applying the six mutation operators defined in Section 5.1.2 to c_i . The process of randomly mutating c_i constructs 20% of the initial population. The rest 80% is comprised of randomly chosen cloud configurations, that are assigned the same service workload as the original configuration c_i. To allow for a diverse population individuals are allowed to differ in length. They may randomly comprise $100 - 1000$ PM chromosomes, corresponding to the cloud capability limits we are exploring in this work (see Section 3.6). The common point among all the initial configuration candidates is the workload assignment. Each individual in the initial population is associated with the same workload as the input configuration c_i. The generation of a diverse initial population with different VM and PM capabilities but with the same workload assignment will create candidates that differ in the non-functional qualities Q_{RT} and Q_E. For example configurations with a small amount of PMs and VMs will have high response times (Q_{RT}), that trades-off with low energy consumption costs (Q_E). On the other side configurations with large cardinality of PMs and VMs will show low response times (Q_{RT}) but high energy costs (Q_E). The diverse initial population will provide a potent starting point for the algorithm to apply the explore the search space while exploiting already promising areas towards converging to an optimal or near-optimal solution.

5.1.2 Genetic Operators

The linear genotype described in section 4.2 permits standard genetic operators, such a single-point crossover, to be used. When a crossover operator is allowed to operate

at any point of the representation, it might be detrimental for the search. The resulting introduction of increased variation to the population can potentially disrupt the breeding process and prevent useful genetic traits from being forwarded to the children [199]. To regulate the amount of disruption we only allow crossover to occur at points between PM chromosomes. More specifically we implement an single-point crossover that allows groups of PMs, each containing one or more VM chromosomes, to be copied to the child being created. The crossover cutting points are selected at random, with respect to the PM chromosomes' boundaries. As a result, the offspring may have variable lengths, enabling the generation of a diverse population. The crossover rate is an important factor as well. A moderate crossover rate is able to achieve a good balance between exploration in the whole search space and exploitation of the specific most promising areas in the search space. Consistent with literature studies that observe that a crossover rate in the range $[0.45, 0.95]$ typically performs well [93], we set it to 80%.

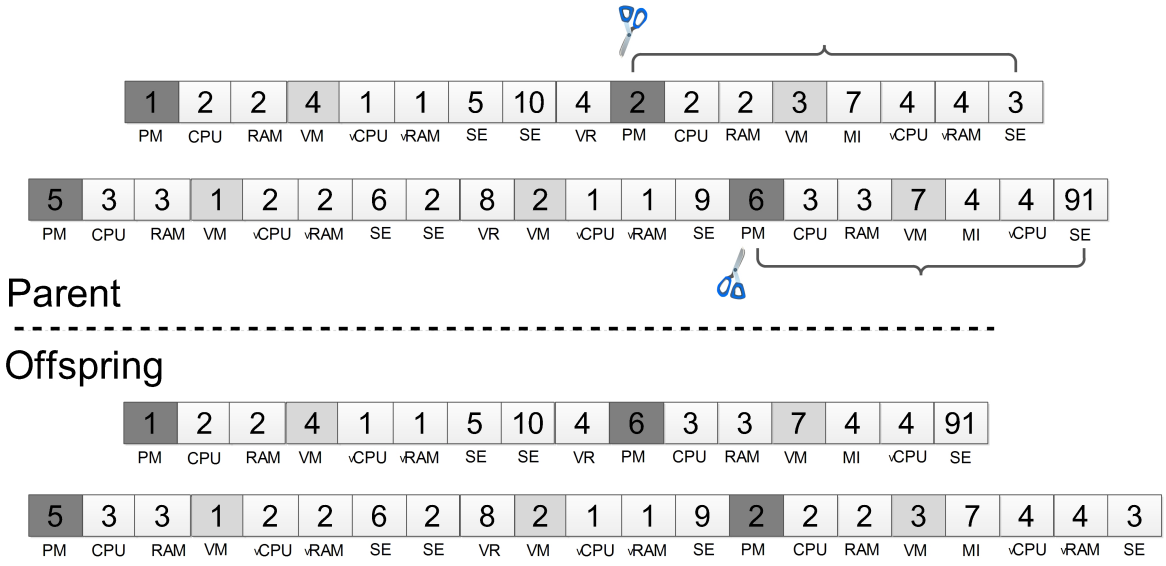


Figure 5.3: Crossover operator example.

Figure 5.3 shows an example of applying the crossover operator. Both parents include two PM chromosomes; the first parent comprises the PMs with ids 1 and 2

while the second parent comprises the PMs with ids 5 and 6. After the crossover has been applied, the PM chromosomes representing the PMs with ids 2 and 6 have been swapped, resulting in two new individuals.

After two parent individuals have produced two children with the help of crossover operator, the mutation operator is applied to each child. In our case while the crossover affected the whole genotype and generated new individuals by swapping PM chromosomes, the mutation operates within the chromosomes' structure. Introducing small modifications to the individuals' chromosomes in a random manner, mutation fosters exploration of so far unexplored genetic material. We have implemented six different mutation operators. The goal of each mutation, is to modify the PMs and VMs along their formalised DoFs (see Table 5.1) and therefore create new feasible configuration solution alternatives. According to our meta-model structure, a VM may be modified along three DoFs; *resize* of vCPU or vRAM, *creation* of a new VM, *termination* of an existing VM, and *migration* of a VM from a source to a destination PM. A PM may be modified along two DoFs; *activate* a new server and *deactivate* an existing one. The available mutation operators are described in the following:

- **M-VR:** Mutates the gene VR (see Table 5.2). Here either the RAM or CPU capabilities of a randomly selected VM are upgraded or downgraded. As a result the mutated VM phenotype will be able to accommodate more or less workload demand than its previous state. As discussed in Chapter 3.6 available ranks for CPU are (500 MIPS, 1000 MIPS, 2000 MIPS) MIPS and for RAM (613 MB, 1.7 GB, 3.75 GB). As a sanity check, a VM cannot be resized if the action exceeds the RAM or CPU capabilities of its host PM.

- **M-VC:** Mutates the number of VM chromosomes. After the mutation is applied, a new VM is created. As discussed in Chapter 3.6, the available VMs configurations simulate the Amazon EC2 high-CPU medium, extra large, small

and micro instances. According to the selected instance size the appropriate genes for vRAM and vCPU are also added to the new VM chromosome. A M-VC mutation also affects the PM chromosome, as the new VM will be mapped in a random PM, with enough free CPU and RAM resources to host it. If no PM has enough computational capabilities to host the new VM, the M-VC action will not be applied. The maximum limit of the VMs' cardinality in our settings is 1000, as discussed in Section 3.6.

- **M-VD:** Mutates the number of VM chromosomes. After the mutation is applied, an existing random VM has been destroyed. Both relevant vCPU and vRAM genes are removed. As a result, CPU and RAM capabilities will be released from the PM, previously hosting the terminated VM.

- **M-MI:** Mutates a VM, mapping it from one PM to another, with enough resources to host the particular VM instance. After the mutation, CPU and RAM capabilities have been released from the source PM, while the VM cardinality in the destination PM have been increased. If no PM has enough computational capabilities to host the migrating VM, the M-MI action will not be applied.

- **M-PA:** Mutates the number of hardware nodes in the cloud configuration, with respect to the user-defined maximum available PMs limit. After the mutation a new PM has been added. The cloud configuration has enforced capabilities and may scale up to serve more demand. The maximum limit of the PMs' cardinality in our settings is 1000, as discussed in Section 3.6.

- **M-PD:** Mutates the number of hardware nodes in the cloud configuration. The mutation operator may be applied only to empty PMs, that do not currently host any VMs. After the mutation, a PM has been switched-off. The cloud configuration has less activated resources and may scale down to save energy costs.

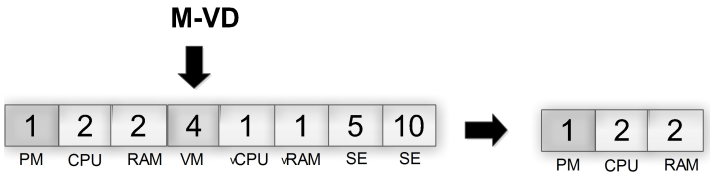


Figure 5.4: Mutation operator example. The M-VD operator removes a VM chromosome from its PM host.

Our mutations always result in coherent new configurations, as there is a sanity check before the application of each action. As described above, the VM mutations are only applied if the underlying host PM has enough available capabilities to support the change in the hosted VM configuration (e.g., increased CPU). Additionally the PM mutations are only applied if maximum limit of 1000 nodes has not been reached.

The above mutations have been designed to generate random diversity to the population by injecting small changes to the individuals. The effect of mutations to the phenotypes correspond to architecture-level reconfigurations. These reconfigurations have a direct effect on the QoS metrics of interest Q_{RT} and Q_E (see Section 3.4) because they may scale up the cloud resources to facilitate the accommodation of service requests and therefore improve the services response time Q_{RT} or release resources enabling energy consumption Q_E savings.

However, such reconfigurations are not free and depending on the reconfiguration action they may introduce performance overheads and/or additional energy costs [114]. VM resizing and VM migration have been the standard tools for online reconfiguration in virtualised environments [192, 193, 152] because they allow on the fly resources adaptation to intensify the exploitation of current datacenter resources. Findings of Verma et al. [195, 194] have shown that model hypervisors implement VM resize with a negligible performance overhead. On the other hand VM migration may affect both energy consumption and the service performance because in requires spare CPU resources to complete. Verma et al. show that VM migration might increase the CPU utilisation of the host server up to 50%. Extra energy is

therefore being consumed in the host PM, due to the linear relationship between CPU utilisation and power consumption [22]. If the required CPU resources are not available, the migration duration will be prolonged. This can increase the downtime of the hosted services and thus degrade the performance of the migrating VM. Finally the actions of adding new VMs and PMs are considered computationally expensive because while they can achieve a more radical effect on improving the datacenter capabilities, the overall datacenter management cost is directly proportional to the number of its computing nodes [194].

Dynamically changing workloads can lead to a complete rethinking of the best strategy for cloud reconfigurations. For example, when the workload is rapidly changing, it may be better to implement cheap but moderate changes than drastic but expensive ones like powering up new hosts, whose costs may never be recouped before new adaptations are required [114]. To avoid flooding the cloud with uncontrolled, costly reconfigurations, we impose priorities on the mutations based on their respective overhead effect. The driving idea is that every problem that occurs should be solved with reconfigurations of the lowest overhead level. Only if this is not possible, the problem is tried to be solved on the next level, and again, if this fails, on the next one, and so on. This prioritisation scheme is aligned with Maurer et al. framework for resolving SLA violations in a cost-efficient manner [144]. The authors divide the possible reconfiguration actions to so-called *escalation levels*. The levels are ordered in a way such that lower levels offer faster and more local solutions than higher ones. ones. The first (lowest) escalation level tries to change the amount of VMs' storage or memory. The next level offers VM migration actions followed by turning PMs on/off at the final escalation level. Similar to our approach, low escalation levels are first attempted to resolve cloud performance problems. If the action does not solve the problem, the next escalation level is explored.

In our case a M-VR mutation will trigger VM resize actions, to examine how changes in VMs' entitlements can manipulate the throughput of existing VMs and therefore affect the overall cloud QoS. If VM resize actions cannot apply or do not suffice to tune performance, VM migrations are explored next by applying the M-MI operator, aiming to release the load from over-utilised PMs and better exploit the capabilities of other, under-utilised active servers. In case the current configuration comprises PMs with over-utilised VMs, requests are queued degrading the services response time while the capabilities of the host PMs are not necessarily fully exploited. We therefore apply the M-VC mutation, that attempt to create new VM instances in active PMs towards alleviating the congestion in neighbouring VM nodes.

In the demand still outgrows the cloud capabilities, new PMs are switched-on with the M-PA operator. VMs are may be deactivated only when they are inactive, i.e., they are not assigned a service resource request. PMs may be switched off when they are idle as well, i.e., they do not host VMs.

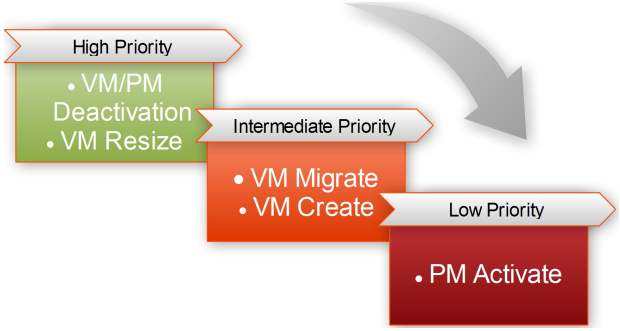


Figure 5.5: Mutation operators priorities.

Our priorities shown graphically in Figure 5.5 are implemented in the form of semi-formal constraints, that seek to apply low overhead mutations first:

S_r denotes a service resource request size, I_{VM} a VM instance type in the available pool of EC2 VM configurations, $>$ stands for demand overload, $<$ stands for underload, $\succ$ contain and H_{PM} a PM hosting one or more VMs.

$$(S_r > VM) \wedge (\exists I_{VM} \succ S_r) \wedge (H_{PM} \succ I_{VM}) \rightarrow I_{VM}(\text{M-VR up}) \tag{5.1}$$

$$S_r < VM \rightarrow I_{VM}(\text{M-VR down}) \tag{5.2}$$

$$(S_r > VM) \wedge (\sum VM > PM) \rightarrow I_{VM}(\text{M-MI}) \tag{5.3}$$

$$(\nexists VM \in PM \succ S_r) \wedge (H_{PM} \succ I_{VM}) \rightarrow I_{VM}(\text{M-VC }) \tag{5.4}$$

$$\nexists S_r \in VM \rightarrow I_{VM}(\text{M-VD}) \tag{5.5}$$

$$\nexists VM \in PM \rightarrow H_{PM}(\text{M-PD}) \tag{5.6}$$

$$\sum PM \not\succ \sum S_r \rightarrow H_{PM}(\text{M-PA}) \tag{5.7}$$

In our simulation setting the BoT workload model (see Section 3.6) models cloud services as black-box resource requests with Weibull arrival patterns, resource request sizes and runtime durations. Each service request is assigned to VMs for execution by the CloudSim VM scheduler [36]; as the service requests compete for the same available capabilities, at each time the services that cannot satisfy their resource needs are queued till VM capabilities become available again.

The constraint (5.1) states that if a service resource request cannot be satisfied by its current host VM, but a new EC2 instance suffices to host the request while the host PM has still enough resources to accommodate this new instance, then the VM will be resized (i.e., scaled up) to the particular instance type (e.g., c3.large). Constraint (5.2) states that if a VM's resources remain underutilised from its hosted services, the VM is scaled down to a smaller EC2 instance. (5.3) states that if both a VM and its host PM are overutilised, the VM is migrated to another PM with enough free capabilities. (5.4) states that a VM is created if no VM in the host PM can serve a request and the PM has enough free resources to accommodate the new VM instance. 5.5 states that an idle VM where services have completed execution

is deactivated. Similarly in 5.6 an idle PM is deactivated. Finally 5.7 states that if the active PMs do not suffice to map the current service request size, a new PM is switched on. Finally we also vary the mutation rates per operator, according to the performance overhead of the corresponding reconfiguration action. Low overhead mutations have higher probability rates. Table 5.3 summarises the selected mutation rates.

Operator	**Rate**
Mutation PA	50%
Mutation PD	80%
Mutation VR	80%
Mutation VC	70%
Mutation VD	80%
Mutation MI	40%

Table 5.3: Mutation rates.

5.1.3 Genotype-to-Phenotype Mapping

The previous section presented the structure of an individual in our generic representation of cloud configuration models. Each of the integers in the individual is responsible for identifying part of the configuration being represented. Once the population of individuals has been created, the next step is to map them to their real-world phenotype representation, to be used by the fitness function. In our problem the real world individual representation corresponds to a CloudSim-based configuration simulation as described in Section 3.6. To map each individual to a simulated configuration we take the following steps:

1. Genetic modifications resulting from mutation and crossover operators, are applied to the genotypes. The mutation and crossover operators result in reconfiguring the initial solution candidates.

2. During mutations and crossovers the individuals are checked for conformance to the structural constraints of our meta-model. Compliance to the meta-model provides a sanity check for the coherence of the new population.

3. Each candidate's chromosomes are mapped to CloudSim entities, to simulate a cloud datacenter. During simulation, the assigned workloads to each VM are processed and at the end of the experiment each configuration is assigned a response time (Q_{RT}) and energy consumption (Q_E) score. The Q_{RT} indicates the time needed to serve the total workload and Q_E measures the energy consumed during this time.

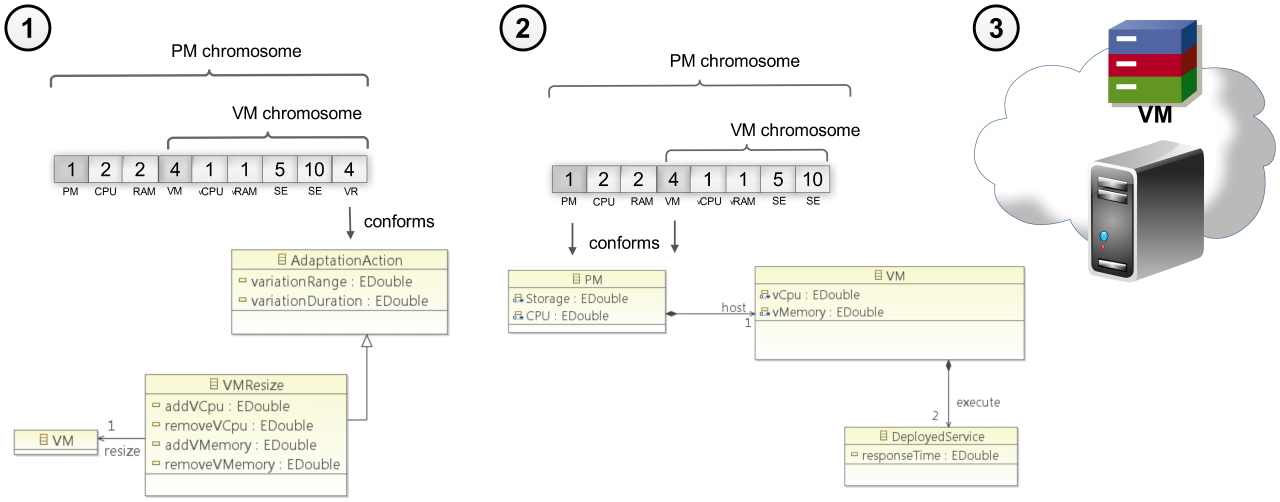


Figure 5.6: The genotype-to-phenotype process. The cloud genotypes are assigned mutation and crossover operators and checked for structural cosistency. Each genotype is mapped to a simulated cloud datacenter using CloudSim. The output of the simulation is the score (Q_{RT} and Q_E) for each candidate.

5.1.4 Fitness Functions

After the individuals have been mapped to their phenotype representation the fitness function can be applied to estimate their fitness scores and differentiate better from worse solutions. The one-to-one correspondence between QoS metrics and fitness functions allows to use the metrics of interest Q_{RT} and Q_E (see Section 3.4)

as fitness functions to guide the search for optimal or near-optimal cloud configurations. In cloud computing as in numerous real world problems, an analytical formula to associate candidate solutions with quality metrics does not exist. Practitioners may use physical experiments to most accurately estimate the quality of possible solutions. However the cost of such experiments is often prohibitively high [112]. Computationally more efficient simulations are preferred to enable the repeatability of experiments [112]. We therefore employ a full system simulation on CloudSim as discussed in Section3.6, to estimate the Q_{RT} and Q_E metrics for each configuration. The measurements achieved for each individual in both metrics constitutes its fitness score.

The use of simulation to measure the candidates' fitness captures is more realistic than employing physical experiments and captures the system behaviour with high fidelity [112], it can still be highly time consuming. The required time to simulate on CloudSim a single candidate solution in the selected population pool of cardinality 100, may rage from $\sim$ 8 seconds to $\sim$ 3 minutes depending on the the assigned workload sizes. As discussed in Section 3.6, in this work we examine workload sizes in the range of $[20000 - 90000]$ BoT tasks. To limit the computation cost of the search we limit the number of GA generations to 30. Using our settings of population size 100 and 30 generations search requires 3000 evaluations to execute. This takes from ~ 7 hours to ~ 7 days to run, depending on the size of the assigned workload

5.1.5 Summary

In this chapter has introduced SBSE, a state of the art approach to optimise software engineering problems. We then defined a representation and appropriate genetic operators to enable the reformulation of the cloud configuration optimisation problem to SBSE. Figure 5.7 summarises the search-based optimisation process discussed in this

Chapter, to explore configuration solutions that best balance the trade-offs between the quality metrics of interest; response time (Q_{RT}) and energy consumption (Q_E) .

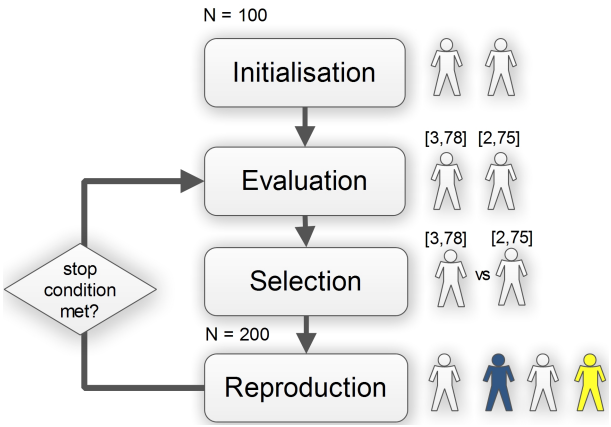


Figure 5.7: Optimisation process.

The first step generates the problem's initial population of cardinality $N = 100$. Each individual is then evaluated using simulation and all candidates are assigned a two-fold fitness score, corresponding to their measurements for Q_{RT} and Q_E. In the selection step the most promising individuals are selected as "parents" for the breeding phase, using tournament operators. In the reproduction step, new solutions are generating based on the previously identified "parents" using crossover and mutation operators. The process iterates till a configurable termination condition has been reached. Commonly, the GA terminates after a number of generations has been reached or after a certain time is elapsed [88]. As discussed in Section 5.1.4, our algorithm terminates when 30 iterations have been reached.

The problem of cloud configuration optimisation complies with the characteristics specified by Clarke et al. [51] to identify good application candidates for SBSE; The search space of all possible configurations is large because of the number of available parameters and adaptation actions. Each machine contains a large number of configurable components such as CPU, memory, VMs and the VMs at each host may interfere with each other competing for the available resources. Additionally, there are no a-priori known or intuitively understood good solutions that could simultaneously

balance the QoS metrics of response time Q_{RT} and energy consumption Q_E. Changes in the dynamic cloud environment such as workload fluctuations affect the underlying optimisation problem and would require a new execution of the metaheuristic search to update the best trade-off configurations according to the current environmental conditions. The one-to-one correspondence of metrics and fitness functions allow a straightforward use of the metrics of interest Q_E and Q_{RT} to guide the optimisation search. However the use of simulation to estimate Q_E and Q_{RT} can be highly time-consuming and may take us from $\sim$ 7 hours to $\sim$ 7 days to run, violating the cloud run-time constraints. The next Chapter describes the use of surrogate models to reduce the search time from hours to seconds.

Chapter 6

Fitness Function Approximation

Search-based software engineering (SBSE) approaches are ideal to explore large complex problem spaces with competing objectives. The ease of reformulating problems as search-based problems and the robustness of the achieved solutions have contributed to the establishment of SBSE as credible optimisation tools with applications ranging from aircraft construction [176], financial forecasting [55] and drug design [127] to to all aspects of the software development life-cycle [12, 68, 199]. However, it is often implicitly assumed that there exists a simple means for evaluating the fitness functions towards measuring and comparing the quality of possible solutions. In practice however, straightforward definitions of analytical fitness evaluation formulas do not exist for the majority of real-world optimisation problems. As a result, full system simulations or physical experiments are required to evaluate the quality of the solution candidates. In Section 5.1.4 we described the use of CloudSim simulations, to measure the QoS criteria Q_{RT} and Q_E of cloud configuration solution candidates. The use of simulation to evaluate a population of 100 candidates over 30 generations result in a cost-prohibitive optimization processes that can take up to 7 days to run.

In this Chapter we address the problem of time-consuming fitness function evaluations towards making our optimisation framework affordable at runtime. Our goal is

to use appropriate statistical regression techniques so-called surrogates, to devise an explicit, analytical formula of the fitness function and substitute the costly simulation-based evaluations. In Section 6.1 we detail different techniques for approximating complex fitness functions. Having introduced in Section 2.1.3 the foundations of statistical regression, we detail in Section 6.2 the process of implementing problem specific surrogate models. Finally section 6.3 presents an overview of the chapter.

6.1 Approximate Fitness Functions

The complexity of the SBSE meta-heuristic algorithm is proportional to the complexity of the fitness function used to evaluate the quality of candidate solutions [112]. As a result experiment-based or simulation-based fitness estimation choices are extremely time consuming and often result in cost-prohibitive optimization processes, which require several hours or even days to converge [111].

As in numerous real world problems, an analytical formula to associate candidate solutions with quality metrics does not exist for the cloud configuration optimisation problem. To estimate the quality metrics "Response Time" Q_{RT} and "Energy Consumption" Q_E that comprise our fitness functions, we employ a full system CloudSim simulation per configuration candidate. While the measured metrics reflect the true system behaviour, the simulation for each single fitness evaluation is timely and may range from ~ 8 seconds to ~ 3 minutes depending on the workload settings. Consequently a single run of our NSGA-II algorithm may require from 7 hours to ~ 7 days as discussed in Section 5.1.4.

An intuitive way to reduce the search time of MOEAs when dealing with expensive fitness functions, is the use of computationally efficient approximation models [134]. Using approximation models the computational burden can be greatly reduced since the efforts required to build and use them are orders of magnitude lower than those

required in a simulation-based fitness evaluation approach. Prominent strategies towards fitness approximations are based on machine learning and may be classified into the following categories [30, 111]:

- **Problem approximation:** The original problem statement is replaced by one which is approximately the same as the original problem but easier to solve.

- **Data-driven functional approximation:** An analytical expression is constructed for the fitness function based on history data observations describing the mapping between the design parameters of solutions and the quality of the design. The approximation models derived from history data traces are often known as *surrogates*. The field of statistical regression offers several techniques for developing efficient surrogates such as Neural Networks [78], Gaussian Processes (also referred to as Kriging models)[92], Support Vector Machines [92].

- **Evolutionary approximation:** This type of approximations aims at reducing the number of required fitness function evaluations by estimating (i.e., imitating) an individual's fitness from other similar individuals. Popular techniques towards fitness imitation are clustering, and instance based learning. Clustering techniques divide candidates into multiple groups according to their similarity. Then only a single individual is selected to represent the fitness of each cluster. Instance based learning methodologies suggest the inheritance of fitness among candidates. These simplistic methods are prone to errors because of the coarse-grained fitness estimations and have been found not to perform well for multi-objective problems [59].

While both data-driven and evolutionary approximations are relevant for reducing the burden of expensive fitness function calls, we decide to focus on surrogate models in our study. Our decision is based on the fact that surrogates make a minimal number of initial hypothesis on the characteristics of the fitness landscape. For example

clustering techniques require in advance specification of the number of clusters while fitness imitation ignores information of new solution candidates. To the contrary, surrogate techniques can accurately "learn" the true fitness function from recorded data, in which the environment features (e.g., workload) and the quality outcomes (Q_{RT}, Q_E) are observed.

6.2 Specifying Surrogates for Fitness Approximation

In this section we will detail the process of developing surrogate models to approximate the QoS metrics of interest in this study Q_{RT} and Q_E. The surrogates will provide a series of different models of the relationship between the cloud configuration predictor variables with the QoS metrics of interest Q_{RT} and Q_E, that constitute our dependent. In Section 6.2.1 we describe the task of selecting variables that can effectively model Q_{RT} and Q_E. To this end we have followed an iterative process of forming assumptions and testing them with graphical methods and test statistics presented in Section 6.2.2, till the most appropriate variables are extracted. Finally in Section 6.2.3 we build a series of linear and non-linear surrogates and comparatively measure their performance towards selecting the most promising models.

6.2.1 Selection of Relevant Variables

Our goal is to select appropriate predictor variables in our cloud scenario that can efficiently explain our QoS criteria of mean response time (Q_{RT}) and energy consumption (Q_E). Towards this purpose there are two conflicting requirements [41]. On the one hand regression models must account for high variation to fit the data closely, which points in the direction for inclusion of large number of variables. On the other hand efficient models must adhere to the principle of parsimony, which suggests the

use of a minimum amount of predictors for ease of understanding and interpretation. Based on the above, our goal is to identify the minimum number of predictors that can sufficiently explain our response variables.

Application performance modelling in virtualised datacenters is a ripe area of research. Several studies have investigated the key virtualisation architecture independent parameters, which influence the performance of a diverse set of applications [58, 139, 154, 186, 155]. Identifying the ideal set of performance influencing parameters in a datacenter, further requires that these are controllable and sufficiently high-level to facilitate comprehension by system administrators and easy manipulation by automated improvement processes. This is particularly relevant for our problem: our predictors must be tunable points for controlling rather than observing the configurations' quality. This way extracting the predictor values during the fitness evaluation process (see Chapter 5) can enable direct estimation of the configurations' quality without the need to e.g. run additional simulations/experiments to observe and get information of the system execution state. For example the CPU allocation is a variable that consists a tunable control point because its modification is independent of the system execution state and maps directly to the system performance [125]. To the contrary CPU utilisation rate is not a tunable point because it is depended on the system execution state and cannot be modified by external processes [125].

Typically the QoS influencing parameters in virtualised environments fall into one of the following categories: (i) *workload*, (ii) *software* and (iii) *hardware* parameters [23]. The workload parameters describe the load imposed to the system of interest. The software parameters describe features of the basic software and finally the hardware parameters describe the physical components of the system.

A common practise in application performance modelling correlates the average or peak CPU utilization of an application and its observed performance [58, 139, 200, 140]. However, CPU utilization is an observable rather than controllable parameter

and therefore it is ill-suited for our problem. Yet, forcing applications within specific CPU utilisation levels is important for manipulating application performance, as access to CPU resources is a critical parameters for the applications QoS. Based on the findings of Kundu et al. [125] we use instead of CPU utilization the percentage CPU allocation which imposes an upper limit on CPU utilization, and is a basic performance control parameter across all virtualization architectures and solutions. The use of memory utilization for modelling application performance has been also explored [181]. Following the same rationale for the CPU, the memory allocation rather than utilisation parameter is better suited for our problem. Other performance influencing factors reported in the literature, include request arrivals rate [48], resource queue lengths [23], disk and network bandwidth usage. Influencing parameters for energy consumption in a virtualised environment typically include the consumption of CPU, memory disk, network interfaces and cooling systems [33].

Based on the literature and our requirement for utilisation independent variables we investigate the following predictors, summarised in Table 6.1. The *Load* predictor models the incoming workload size and for our problem corresponds to the size of the submitted BoT (i.e., total jobs number). The *Request size* represents the average CPU request in MIPS. The predictors *PMs* and *VMs* denote the total number of active hardware and virtual nodes in the cloud respectively. PMs in our case study are multi-core and may combine from $2-6$ processing units each. The processing cores are also heterogeneous with frequency within the range $[1860, 2660, 2933, 3076]$ MIPS. The variable *Cores* models the total number of processors in the cloud and the *Processor speed* models the average core processing power in the system. The variable *Allocated CPU* denotes the percentage of CPU allocation in the cloud. The allocated CPU indicates the maximum level of CPU utilisation that can be achieved using the current system settings. The allocation percentage is the ratio of the sum of MIPS corresponding to the virtual CPU capabilities of the total active VMs, to

the sum of MIPS corresponding to the CPU capabilities of the total active PMs. The equation 6.1 shows the CPU Allocation for M VMs and N PMs.

$$\frac{\sum_{i=1}^{M} vCPU_i}{\sum_{j=1}^{N} CPU_j} \tag{6.1}$$

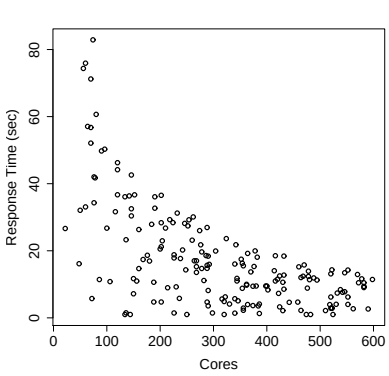
As the VMs act as a container for the incoming services to execute, the sum of virtual CPU capabilities define the maximum amount of physical CPU that may be utilised by the services. In a cloud datacenter of many PMs and a single VM for example, the incoming services can only utilise the virtual CPU of the particular VM, and cannot access additional resources unless more VMs are activated.

Similarly the variable *Allocated RAM* is the ratio of the sum of MB corresponding to the RAM capabilities of the total active VMs, to the sum of MB corresponding to the RAM capabilities of the total active PMs.

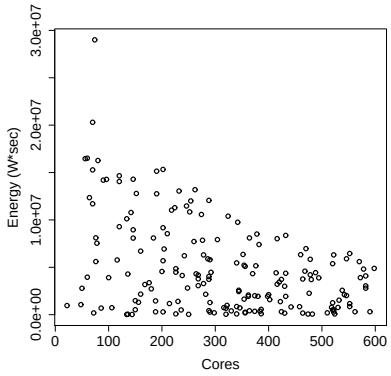
Possible Predictor	**Description**
Load	Total submitted number of resource requests in the cloud since the beginning of time
Request size	Average CPU demand per resource request
PMs	Total number of Physical Machines in the cloud configuration
Cores	Total number of processing units in the cloud configuration
Processor speed	Average processor speed in the cloud configuration
VMs	Total number of Virtual Machines in the cloud configuration
Allocated CPU	Upper limit on CPU utilisation of the cloud configuration
Allocated RAM	Upper limit on the RAM utilisation of the cloud configuration

Table 6.1: Potential predictor variables. The predictor values will vary among different cloud configurations and explain variations in the metrics of interest Q_{RT} and Q_E accordingly.

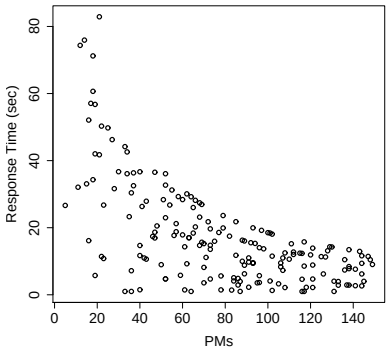
After the identification of potentially relevant factors to explain our responses Q_{RT} and Q_E summarised in Table 6.1, we proceed with collecting data from our environment to begin the analysis.



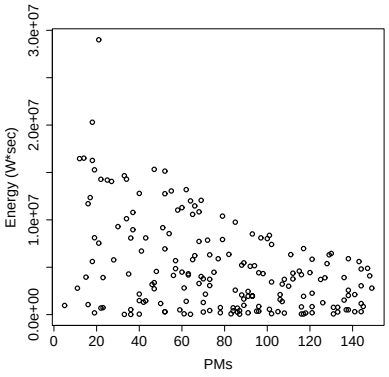
(a) Cores vs response time



(b) Cores vs energy



(c) PMs vs response time



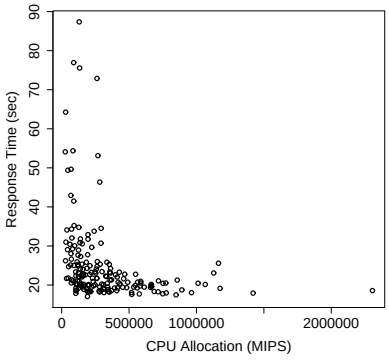
(d) PMs vs energy

Figure 6.1: Scatter plots of predictors cores and PMs versus the responses energy Q_E and response time Q_{RT}

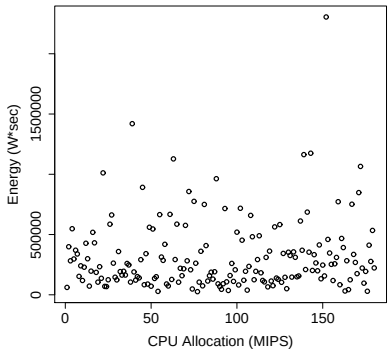
6.2.2 Data Collection

Large data samples tend to result in close to zero p-values A.2.1. This effect may lead scientists to claim results of no practical significance [135, 11]. Published sample estimation tables recommend a maximum threshold of 1111 observations [109, 173, 206] considering 95% confidence level. We therefore uniformly sample our parameter space described in 3.6 to collect 1000 configurations.

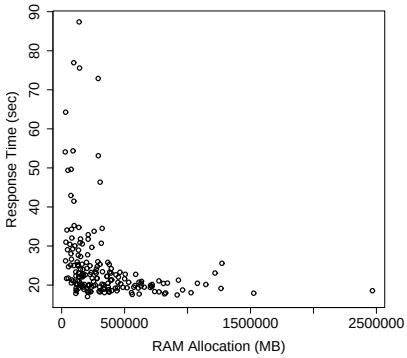
Before collecting the full data set, we assess the usefulness of our predictors to explain Q_{RT}, Q_E by examining their correlations A.2 and corresponding scatter plots on an initial sample of 200 observations. Our goal is to eliminate irrelevant explanatory variables at an early stage to save time and resources.



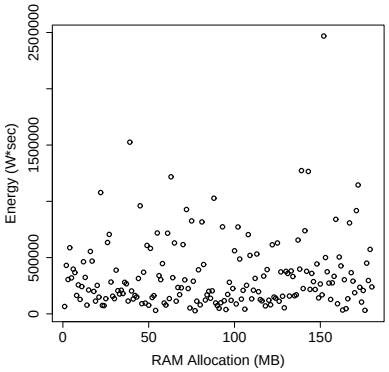
(a) CPU allocation vs response time.



(b) CPU allocation vs energy



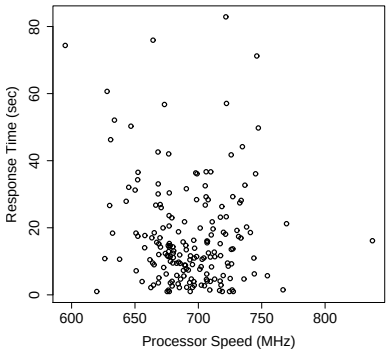
(c) RAM allocation vs response time



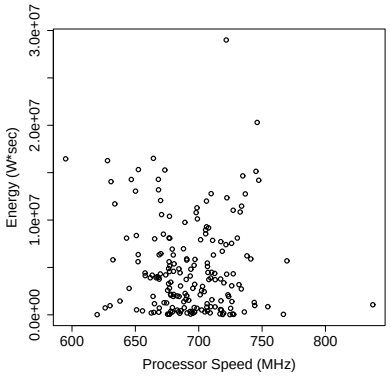
(d) RAM allocation vs energy

Figure 6.2: Scatter plots of predictors CPU and RAM allocation versus the responses energy Q_E and response time Q_{RT}

Figure 6.1 illustrates pairwise plots of the predictor variables cores and PMs versus the responses. We observe that the inclusion of both variables is redundant as they appear to add the same amount of information towards explaining Q_{RT} and Q_E. Cores and PMs appear to be highly correlated with correlation of 0.995, which verifies our previous point, that they add redundant information to the model. Similarly CPU and RAM allocation shown in Figure 6.2 have correlation of 0.999. Finally we observe in Figure 6.3 that the processor speed develops no clear pattern with neither response, measuring correlation of -0.111 and -0.06 with Q_{RT} and Q_E respectively.



(a) Processor speed vs response time



(b) Processor speed vs energy

Figure 6.3: Scatter plots of predictor processor speed versus the responses energy Q_E and response time Q_{RT}

After this initial measurement of predictors' usefulness we proceed with collecting the full dataset of 1000 observations considering only the most influential predictors, as summarised in Table 6.2.

Predictor	**Description**
Load	Total submitted number of resource requests
Request size	Average CPU demand per request
PMs	Total number of Physical Machines in the cloud
VMs	Total number of Virtual Machines in the cloud
Allocated CPU	Upper limit on CPU utilisation of the cloud

Table 6.2: Reduced predictor variables.

Figures 6.4 and 6.5 show pairwise scatter plots between the predictors and our response variables response time and energy respectively. The numerical values inside the grid boxes are highlighted with red stars and provide the corresponding measurements for correlations; for example the response time has correlation 0.19 with load, no correlation with PMs, 0.85 correlation with VMs etc. Finally in the diagonal we observe the distributions of recorded data for each variable of interest.

As discussed in Section 2.1.4, some regression models make strong assumptions about the structure of the data, which often fail to hold in practice such as the

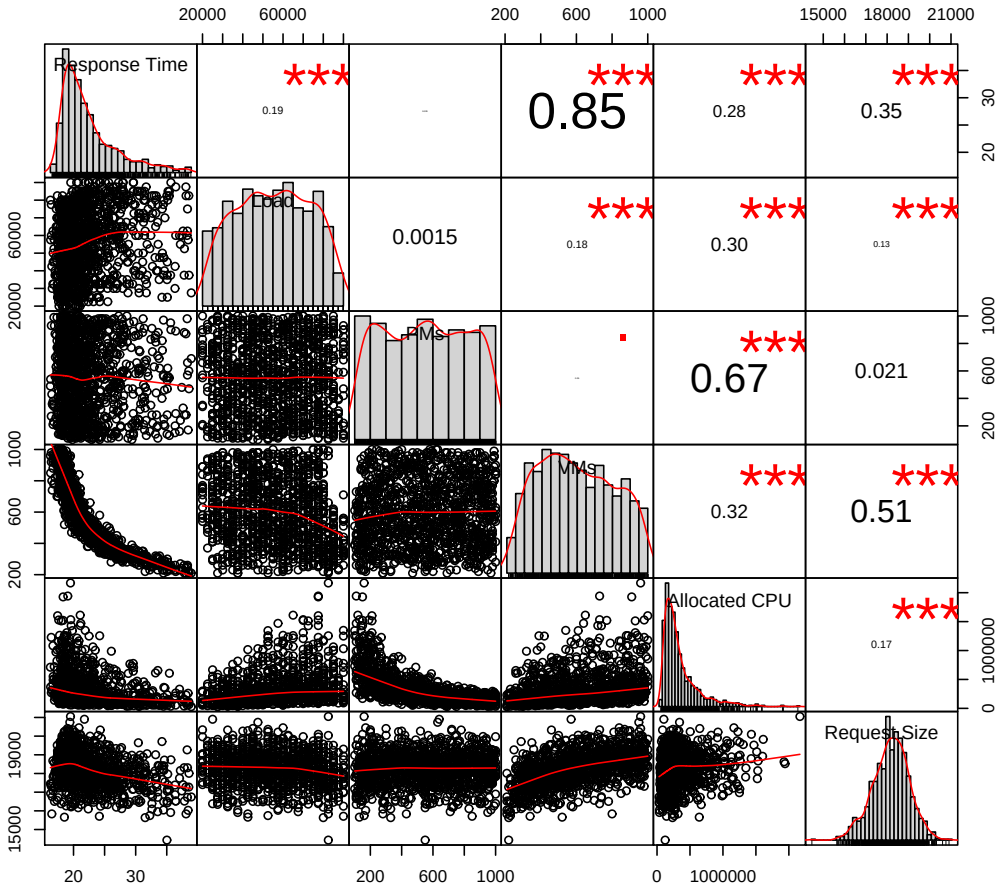


Figure 6.4: Pair-wise scatter plots and correlation measures between the response variable Q_{RT} and the predictors in the recorded dataset. The nummerical values inside the grid boxes, highlighted under the three red stars notion, show the correlation values. The diagonal boxes of the grid show the data distributions of the studied variables. From left to right in the diagonal we see the data distribution of the dependent variable; i.e., *response time* and predictors i.e., *load*, *PMs*, *VMS*, *allocated CPU* and *resource size*. All other boxes in the grid depict pair-wise relationships between the response and the predictors and among the predictors themselfs. The Y axis corresponds to the variable described in the diagonal box of the same **row** and the X axis corresponds to the variable described in the diagonal box of the same **column**. The grid boxes below the diagonal show the graphical relationships between the corresponding Y and X variables. The grid boxes above the diagonal show the correlation values between the corresponding Y and X variables. For example in the first box of the second from the top row show the function between *load* (Y axis) and *response time* (X axis). In the second box of the top row the correlation value between *load* and *response time* is shown.

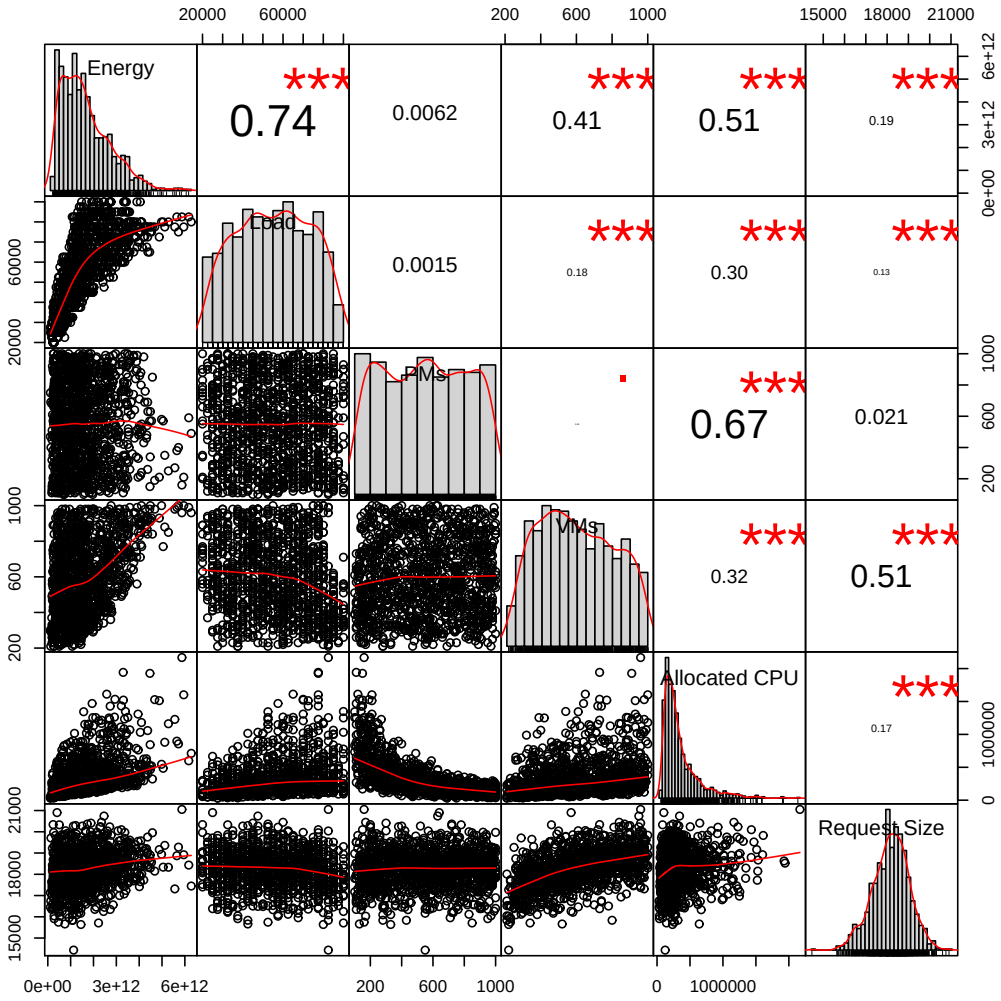


Figure 6.5: Pair-wise scatter plots and correlation measures between the response variable Q_E and predictors i.e., *load*, *PMs*, *VMS*, *allocated CPU* and *resource size* in the recorded dataset.

normality of errors assumption in linear models. Violations of normality may rise because the distributions of predictors and/or responses are themselves significantly non-normal or because the linearity assumption is violated. Non-linear transformations serve many functions in quantitative data analysis, including improvement of the normality of distributions, equalisation of variance and preparing the data for modelling by reducing the influence of unusual variable values. Zimmerman [213] has further pointed out that normalising the data is not only relevant for parametric

regression, which makes strong assumptions about the data structure, but also non parametric regression models, where no explicit assumption of normality is made, tend to benefit as well.

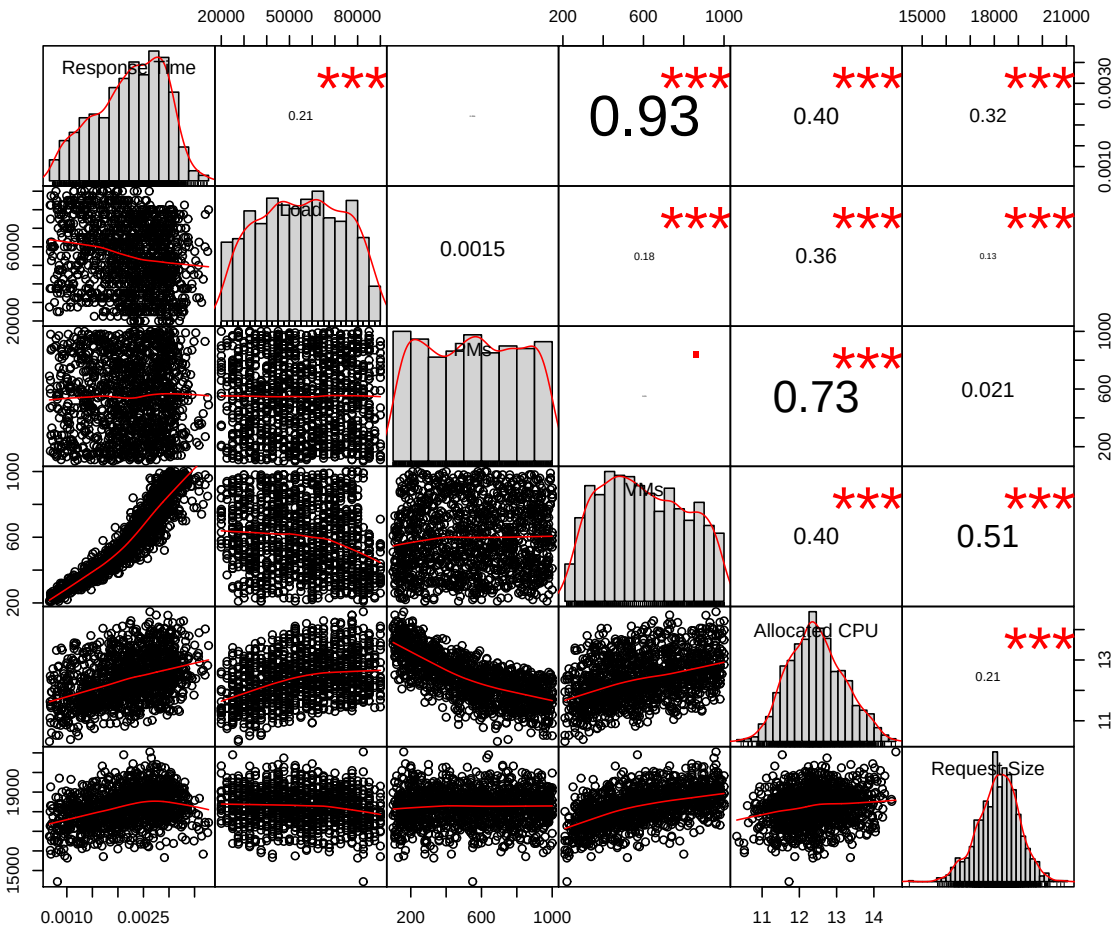


Figure 6.6: Pair-wise scatter plots and correlation measures between the response Q_{RT} and predictors i.e., *load*, *PMs*, *VMS*, *allocated CPU* and *resource size* in the transformed dataset.

We therefore pre-process the recorded dataset by applying appropriate non-linear transformations. A log transform corrected here the right skewness in the predictor "Allocated CPU" and in response "Energy", while a negative power transformation compressed large values in the response "Response Time" and spread out larger ones [73]. The predictors "Load" and "VMs" approximately resemble a normal distribution

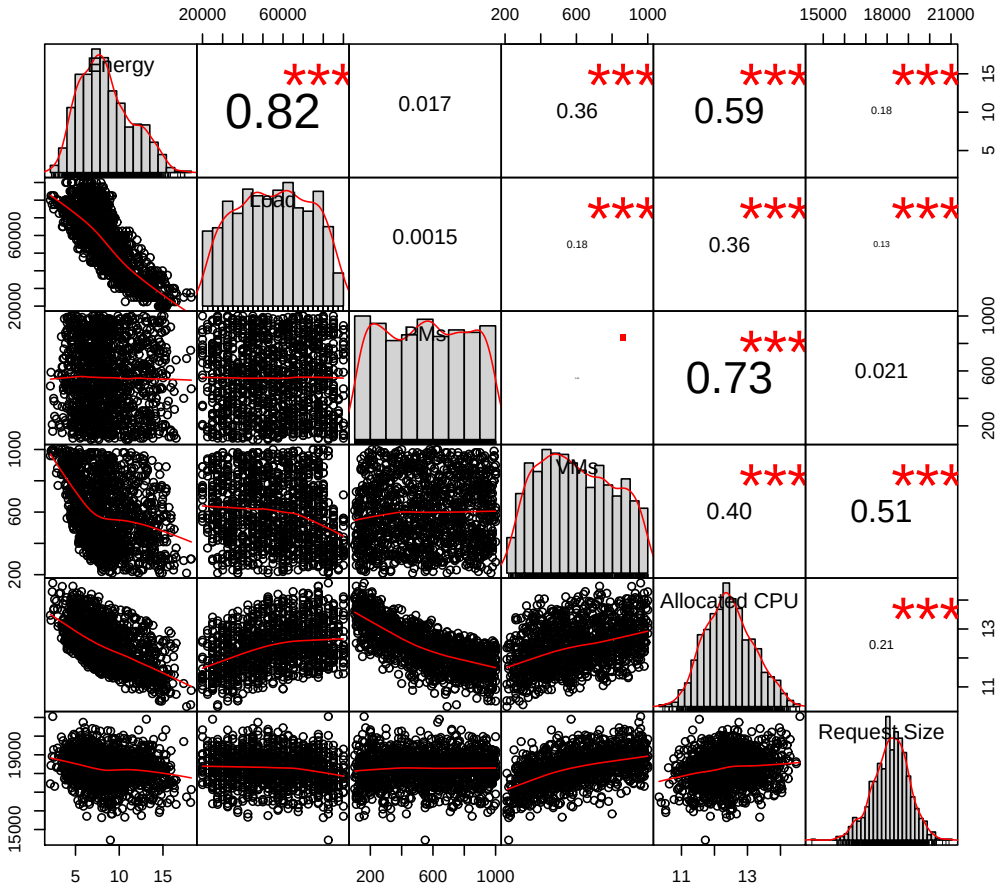


Figure 6.7: Pair-wise scatter plots and correlation measures between the response Q_E and predictors i.e., *load*, *PMs*, *VMS*, *allocated CPU* and *resource size* in the transformed dataset.

while no transformation managed to improve the shape of "PMs". Figures 6.6 and 6.7 show the new scatter plots and correlations. We observe an overall improvement of correlations between responses and predictors after transforming the recorded data. Regarding "Response Time", its correlation with "Load" improved from 0.19 to 0.21, VMs from 0.85 to 0.93 and CPU 0.28 to 0.4. Similarly, the correlations between "Energy" and "Load" improved from 0.74 to 0.82 and CPU from 0.51 to 0.59. Our analysis shows that the predictor "PMs" is not influential as it does not develop clear patterns with neither response variable. This point is also confirmed by the

corresponding correlations, which approach zero. As a result, the predictor "PMs" is also eliminated. Table 6.3 summarises the finally considered predictors.

ID	Predictor	Description
Load	Load	Total submitted number of resource requests
Reqsize	Request size	Average CPU demand per request
Vms	VMs	Total number of Virtual Machines in the cloud
Cpualloc	Allocated CPU	Upper limit on CPU utilisation of the cloud

Table 6.3: Final predictor variables.

Based on the identified predictors the surrogates $S_{i_{RT}}$ and S_{i_E} to predict Q_{RT} and Q_E respectively will take the following forms:

$$S_{i_{RT}} = f_i(load, reqsize, vms, cpualloc) \tag{6.2}$$

$$S_{i_E} = g_i(load, reqsize, vms, cpualloc) \tag{6.3}$$

Training Data

Having identified the most influencing variables to explain Q_{RT} and Q_E we proceed to collecting the training dataset. Each dataset entry describes the relation between a configurations' structure with respect to the predictors (load, request size, VMs and allocated CPU) and its corresponding quality with respect to the responses Q_{RT} and Q_E. The data reflect the cloud operation in a *steady-state*, which means that the neither the workload nor the datacenter capabilities (e.g., CPU, RAM) are changing during this time. Therefore the system structure and quality measurements are consistent.

The cloud system is simulated on CloudSim as discussed in Section 3.6. As mentioned in Section 3.6, the studied cloud settings in this work span BoT workloads of size $20000 - 90000$ and VMs scale within $100 - 1000$. Table 6.4 summarises the

parameter ranges used to train the models. The training ranges are extracted by uniform sampling within the use case parameter limits.

ID	Predictor	Training Range
Load	Load	20000 – 90000 BoT jobs
Reqsize	Request size	14433 – 21051 MIPS
Vms	VMs	100 – 1000
Cpualloc	Allocated CPU	0 – 100% MIPS

Table 6.4: Training data ranges.

6.2.3 Surrogate Models Description

So far we have presented the selection process of influential variables to explain the Response Time (Q_{RT}) and Energy Consumption (Q_E) in cloud configurations. We then collected data from our simulation environment (see 3.6) consisting of 1000 observations on the specified 4 predictors 6.3 and our 2 responses; Q_{RT} and Q_E. The next step is to identify the form of the models $\hat{Q_{RT}}$ and $\hat{Q_E}$ that will relate each of our responses to the set of predictors in the form of a multiple regression equation. For this purpose we will devise and asses the predictive ability of a set of diverse models presented in Section 2.1.4. All models are implemented in the open source statistics tool R [4].

Linear Surrogates

First we use linear regression to model our response variables "Response Time" Q_{RT} and "Energy"' Q_E, which is the most widely used statistical technique describing the simplest possible, non-trivial relationship among the variables of interest. The resulting models $\hat{Q_{RT}}$ and $\hat{Q_E}$ are described by the equations 6.4 and 6.5 respectively. Tables6.5 and 6.6 show the regression outputs for each model, summarising the coef-

ficient values, standard errors, t-values and p-values, estimated using the R function $lm()$ [4] for linear regression.

$$\hat{Q_{RT}} = \beta_0 + \beta_1 Load + \beta_2 Vms + \beta_3 Cpualloc + \beta_4 Reqsize \tag{6.4}$$

$$\hat{Q_E} = \beta'_0 + \beta'_1 Load + \beta'_2 Vms + \beta'_3 Cpualloc + \beta'_4 Reqsize \tag{6.5}$$

Variable	**Coefficient**	**Standard Error**	**t-value**	**p-value**
Constant	$2.742e-03$	$1.740e-04$	15.759	$< 2e-16$
Load	$-2.799e-09$	$3.869e-10$	-7.234	$8.27e-13$
Vms	$3.264e-06$	$3.812e-08$	85.634	$< 2e-16$
Cpualloc	$6.209e-05$	$1.007e-05$	6.165	$9.61e-10$
Reqsize	$-1.692e-07$	$8.135e-09$	-20.805	$< 2e-16$

Table 6.5: Regression Output for $\hat{Q_{RT}}$.

Variable	**Coefficient**	**Standard Error**	**t-value**	**p-value**
Constant	$2.653e+01$	$6.459e-01$	41.076	$< 2e-16$
Load	$-1.486e-04$	$1.436e-06$	-103.437	$< 2e-16$
Vms	$-7.022e-03$	$1.415e-04$	-49.625	$< 2e-16$
Cpualloc	$-2.960e-01$	$3.739e-02$	-7.916	$5.49e-15$
Reqsize	$-9.822e-05$	$3.020e-05$	-3.253	0.00118

Table 6.6: Regression Output for $\hat{Q_E}$.

The p-values in Tables 6.5 and 6.6 are much less than 0.05 indicating that all estimated coefficients are highly significant. However, the models cannot be considered valid if any of the standard linear regression assumptions 2.1.4 are violated by the fitted data. Figures 6.8 and 6.9 summarise diagnostic tests on the linear regression assumptions for $\hat{Q_{RT}}$ and $\hat{Q_E}$ respectively. The normal Q-Q plot in Figure 6.9 shows that the points deviate from the reference line, indicating that the residual errors for $\hat{Q_E}$ are not normally distributed. For both models the *Residuals vs Fitted* plot

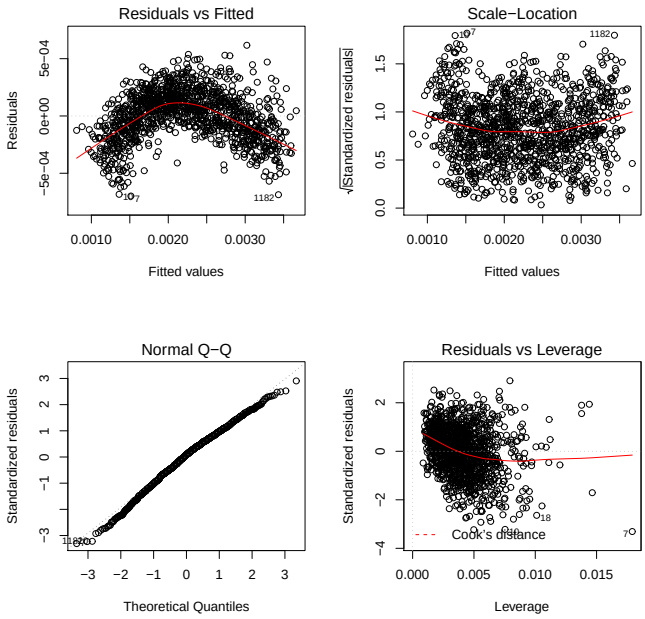


Figure 6.8: Diagnostic checks for Response Time.

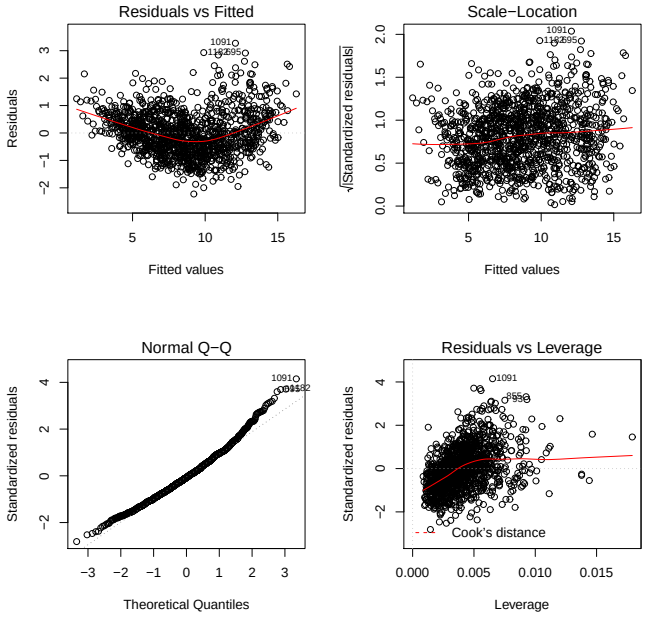


Figure 6.9: Diagnostic checks for Energy.

define a parabolic plot which indicates that the errors violate the the homoscedasticity assumption. Furthermore, for both models the *Scale-Location* and *Residuals vs Leverage* plots show points that are scattered away from the center, which suggests

that some observation have excessive leverage. The presence of leverage indicates that the models have failed to capture important data characteristics. The diagnostics showed that linear models do not suffice to explain our response variables, hence we reject both models described in 6.4 and 6.5.

Non-Linear Surrogates

When building non-linear regression models on R, a set of parameters must be tuned. In particular for the CART model, the parameter *minisplit* specifies the required minimum number of observations in a tree node to split a new leaf, and *cp* is an indicator of the acceptable tree complexity. For the MARS model, the parameter *nk* is the maximum number of hinge functions, *thresh* is a termination criterion for the pruning process, and *nprune* the maximum number of hinge functions after pruning. For the SVR model, the parameter *kernel* stands for the kernel function, *gamma* specifies kernel parameters, and cost defines a penalty for the model complexity. Finally for the GPs model the parameter *kernel* is the covariance function.

The goal is to find the optimal parameter settings for the considered regression models. The choice of the optimal parameters is decided here by an exhaustive search on the possible combinations of parameters in the range shown in Table 6.7. The range of explored parameter values has been based on common practice choices [183, 153, 148, 54, 7]. The search process finally selects the parameter combinations which reduce the Root Mean Squared Error (RMSE) of the studied model.

The next step after parameter tuning for the surrogates, is the identification of the best subset of predictor variables that are sufficient for explaining their joint effect on the response variables studied in this problem; Q_{RT} and Q_E. This step is part of the model selection process (see Section 2.1.4), which aims at trading bias off with variance in a way that performance metrics, detailed in Section 2.1.4, *(i.e., R^2, MAE, MSE, RMSE, MAPE)* are optimised. Apart from balancing the bias vs variance trade

Model	Parameters	Tested Range	Used Values
CART	*minisplit*	$[10, 100]$	50
	cp	$[0, 1]$	0.01
MARS	*nk*	$[5, 100]$	6
	thresh	$[0.001, 0.005]$	0.005
	nprune	$[1, 80]$	5
SVR	*kernel*	[polynomial,radial basis,sigmoid]	radial basis
	gamma	$[1e-6, 0.1]$	0.1
	cost	$[0.1, 10]$	10
GPs	*kernel*	[radial basis, hyperbolic, laplacian]	radial basis

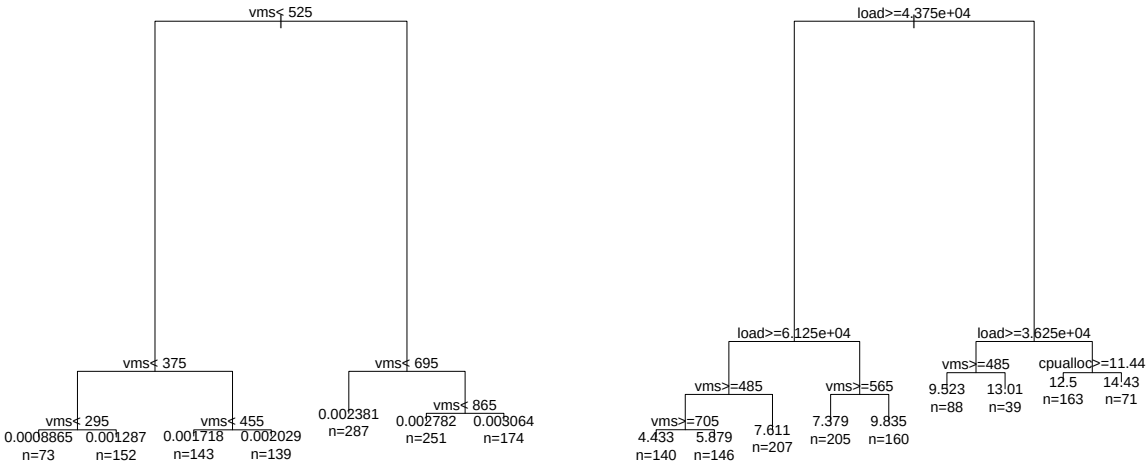
Table 6.7: Examined Model Parameters

off, the selection of most influential subset of predictors increases the interpretability of the models.

The CART models on R were constructed using the function *rpart*() [4, 183]. The integrated model selection process in *rpart*(), identified "VMs" (*vms*) as the only significant predictor for modelling Q_{RT}. The corresponding model is shown in Figure 6.10a. In a similar fashion, the CART for Q_E used the predictors "Load" (*load*) , "Request size" (*reqsize*) and "Allocated CPU " (*cpualloc*). The corresponding tree for modelling Q_E is shown in Figure 6.10b.

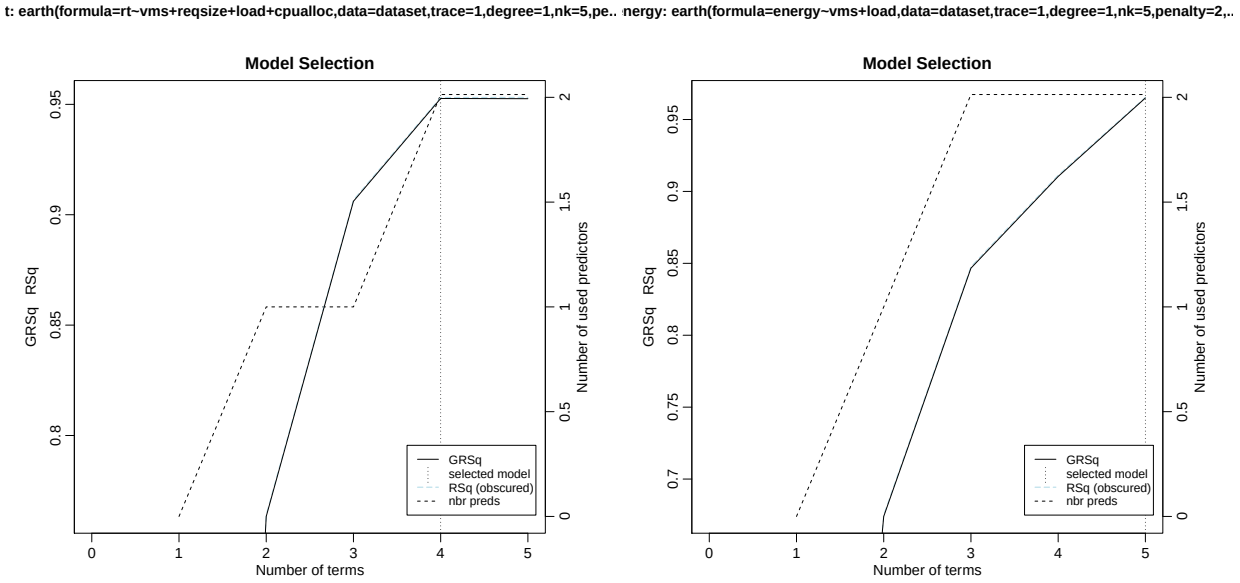
For the construction of the MARS models on R we used the function *earth*() [4, 148]. The integrated model selection process in *earth*(), identified *vms* and *reqsize* as significant predictors for modelling Q_{RT}. Figure 6.11a shows the corresponding model selection process, illustrating that two out of the total four predictors were selected and four hinge functions remained in the final model after pruning. The MARS for Q_E included two predictors as well; *vms* and *load* while five hinge functions remained in the final model as Figure 6.11b shows.

For the construction of the GP models on R we used the function *gausspr*() from the package *kernlab* [4, 7]. As there is no integrated model selection function here, we applied a custom *backward elimination procedure*. Beginning with a full equation, variables are dropped on the basis of their contribution to the reduction of



(a) CART model for Response Time Q_{RT}. (b) CART model for Response Time Q_E.

Figure 6.10: 3-dimensional scatter view of the recorded transformed data.



(a) MARS model selection for Q_{RT}. (b) MARS model selection for Q_E.

Figure 6.11: MARS models selection.

performance errors. The selected GP model for Q_{RT} used a subset of three predictors (vms, $cpualloc$, $reqsize$). For Q_E three predictors ($load$, vms, $cpualloc$) were selected.

Finally, for the construction of the SVR models on R we used the function $svm()$ from the package *kernlab* [4, 7]. The selected SVR model for Q_{RT} and Q_E included the full predictor set. Figure 6.12 summarises the overall process.

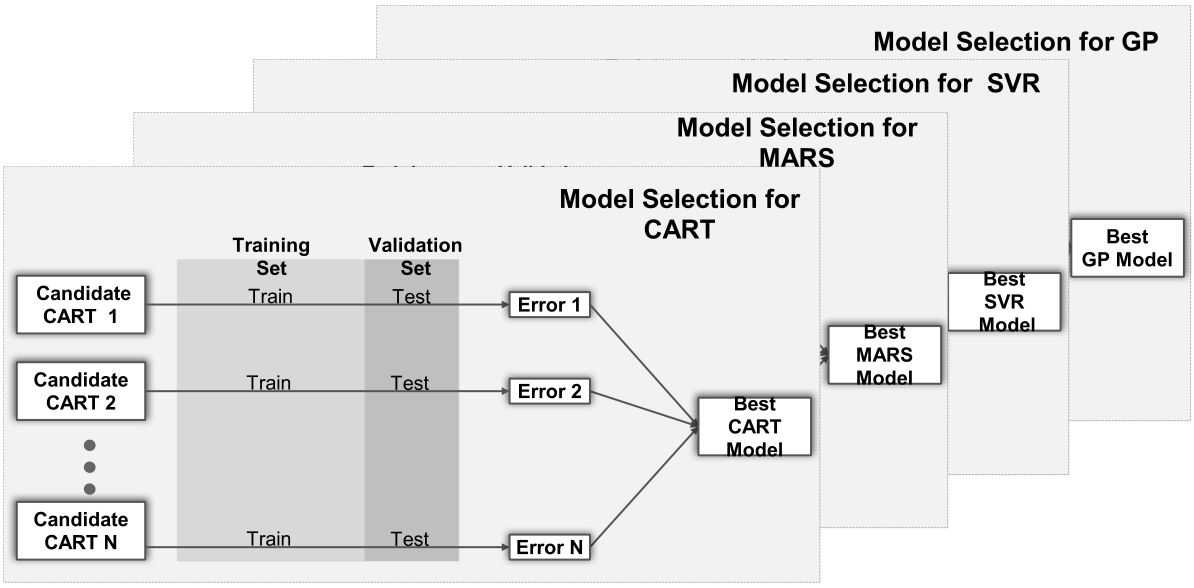


Figure 6.12: Identifying surrogates with sufficient predictor subsets [64].

6.2.4 Surrogate Models Selection

In Section 6.2.3 we described the devised regression models to approximate the responses Q_{RT} and Q_E. The first part of the model selection process tuned the different regression model parameters and complexity to optimise their prediction accuracy based on the performance metrics R^2, MAE, MSE, RMSE and MAPE as shown in Figure 6.12. The next step, illustrated in Figure 6.13, is to compare the predictive ability of the tuned regression models, towards selecting those that best reflect the real Q_{RT} and Q_E behaviour. Table 6.8 summarises the predictive ability of each model using 10 fold cross-validation on the recorded dataset. The technique starts by randomly partitioning the initial data set in 10 subsets. Once the subsets are

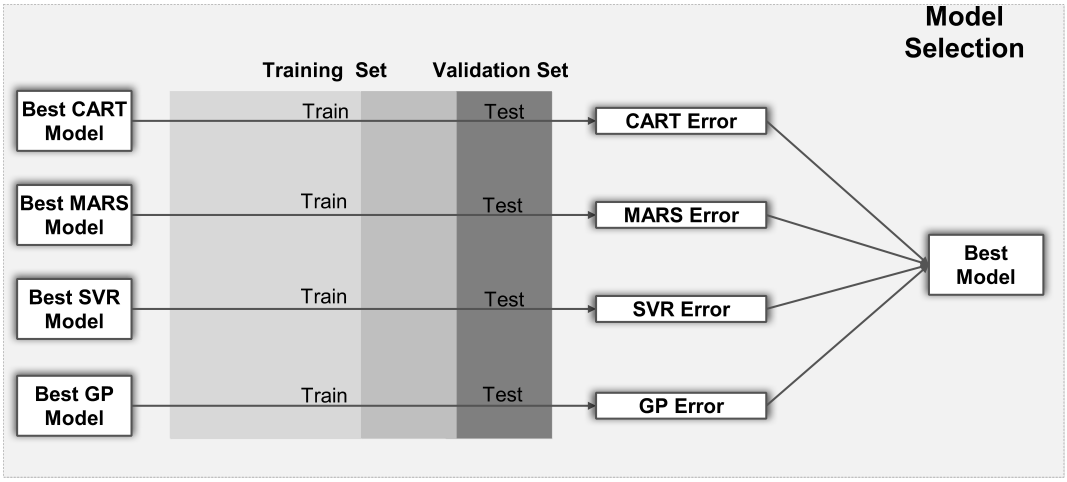


Figure 6.13: Identifying surrogates with sufficient predictor subsets [64].

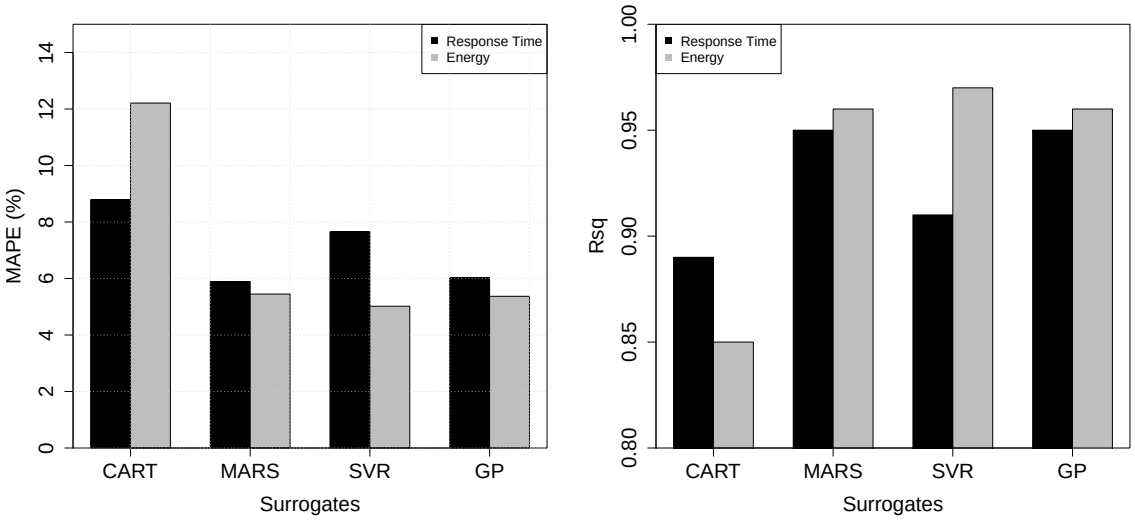
created, it proceeds by iterating the following activities for all the subsets i where $i = 1, \ldots, 10$: (i) Remove from the original dataset the chosen subset i ; (ii) train the model with the remaining data; and, (iii) calculate the prediction error using the subset i. At the end of the process, the prediction error is aggregated over the 10 folds.

Figure 6.14 visualises the difference in performance among the models, using the $MAPE$ and R^2 metrics.

QoS	**Model**	**Accuracy Metrics**			
		R^2	RMSE	MAE	MAPE
Response Time (Q_{RT})	CART	0.89	0.0002	0.0001	8.78%
	MARS	0.95	0.0001	0.0001	5.89%
	SVR	0.91	0.0002	0.0001	7.65%
	GPs	0.95	0.0001	0.0001	6.02%
Energy (Q_E)	CART	0.85	1.19	0.95	12.21%
	MARS	0.96	0.57	0.44	5.45%
	SVR	0.97	0.51	0.39	5.02%
	GPs	0.96	0.56	0.42	5.37%

Table 6.8

Overall the surrogates appear to perform very well. MARS and GP seem to exhibit the best performance for predicting Q_{RT} with 0.95 R^2 and approximately 6%



(a) MAPE measures.

(b) R^2 measures.

Figure 6.14: Prediction accuracy for the surrogates.

MAPE. On the other hand, SVR seems to exhibit the best performance for predicting Q_E with 0.97 R^2 and approximately 5% MAPE.

6.3 Summary

In this chapter we have discussed the importance of fitness function approximations to reduce the burden of computationally expensive cloud configuration candidates evaluation. After a brief overview on statistical regression techniques we presented a series of surrogate models to approximate the quality metrics "Response Time" Q_{RT} and "Energy Consumption" Q_E, which define our fitness function. We have used a diverse range of regression techniques to develop our surrogates ranging from the highly interpretable CART and MARS to the more complex SVR and GPs.

Figure 6.15 summarises the model construction process. Overall, the devised surrogates demonstrated good predictive ability based on 10-fold cross validation results on the recorded dataset. Therefore all statistical models with the exception

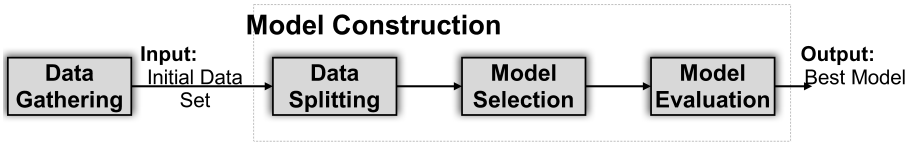


Figure 6.15: Model construction overview.

of the linear model, were accepted at this point. The final model selection will to be based on independent data and follows on Chapter 7.

Chapter 7

Evaluation Study

This thesis has presented a methodology for improving the quality of cloud IaaS configurations, considering the quality metrics Q_{RT}, that stands for the hosted services response time, and Q_E, that stands for the cloud datacenter energy consumption. The previous Chapters 5 and 6 have presented the building components of our approach, namely (i) the SBSE reformulation of the cloud optimisation problem to explore the cloud configuration search space of optimal or near-optimal solutions and (ii) the development of data-driven surrogate approximations to speed-up the search space exploration. Up to this point there has been little discussion about the quality of the optimised configuration outputs of our proposed framework. In this chapter we validate the hypothesis that our approach can optimise cloud configurations at runtime. The validation is structured around three main goals: (i) to assess the efficiency of the cloud configuration optimisation, (ii) to assess the performance of the surrogate models as fitness functions in the optimisation process in terms of their prediction accuracy and timeliness and (iii) to to highlight the strengths and weaknesses of our proposed methodology. The work presented in this chapter has been published in [42, 44].

7.1 Research Questions

To validate our optimisation approach we seek to answer a series of Research Questions (RQs), presented below.

RQ1 *Is the use of SBSE application effective for the optimisation of cloud configurations ?*

The goal of this RQ is to demonstrate that the use of SBSE for optimising cloud configurations is robust and can lead to high-quality solutions. Towards proving this point we will compare the performance of the multi-objective optimisation genetic algorithm, applied to explore the cloud configuration search space with purely random search. According to SBSE practices [91, 90, 89] the performance of the genetic algorithm must comfortably outperform a random search for improved configurations, to demonstrate that the cloud configurations optimisation problem is not trivial and that our search-based approach is indeed effective.

This point also provides a sanity check for the parameter choices in the configuration of the genetic algorithm; while the genetic algorithm is randomised, it is still able to smartly balance the exploration vs exploitation trade-off and introduce biases towards promising areas of the search space. To study this RQ, we use a simulation-based fitness function to measure the quality metrics of interest Q_{RT} and Q_E on CloudSim. We will refer to the simulation-based fitness function as "expensive" fitness function as well, due to the high computation cost required to evaluate the quality of the population.

RQ2 *Are the devised surrogates accurate with respect to their predictive ability?*

Our methodology develops lightweight surrogate models to substitute the initial, expensive simulation-based fitness function with an analytic formula. The models receive as input data on a series of predictor variables (see Table 6.2) that describe a configuration's structure, and yield estimations of the expected fitness of the configuration. The accuracy of the estimations is of paramount importance to the meta-

heuristic search. Inaccurate predictions may lead the search to low-quality solutions and therefore end up wasting system resources or interfere with the services performance. The goal of this question is to assess the prediction accuracy of the devised models and test if they are able to maintain a reasonable level of quality even when noise and unexpected workloads are introduced to the system.

Evaluating the robust performance of our surrogates under unexpected conditions goes beyond our study in Section 6.2.4. In Chapter 6 we presented an *interpolation scenario* i.e., we have used our training dataset of 1000 samples described in Section 6.2.2 to estimate a set of performance metrics for each surrogate. More specifically using 10-fold cross validation, the original dataset is iteratively (for fold $i = 1, \ldots, 10$) split in the training set and the validation set; the first is used to fit the models while the latter is used to compute the R^2, $RMSE$, MAE and $MAPE$ metrics. The results per iteration are combined to give us the final performance estimations described in Section 6.2.2.

Overall, the interpolation scenario evaluates the performance of our surrogates under controlled, anticipated conditions. However, in real world systems parameters such as the workload intensity cannot be controlled, and may vary outside the original training dataset range. In this RQ we introduce *extrapolation* scenarios, where we change the workload intensity and variability outside the training values range. Our goal is to prove that the models remain sufficiently accurate and can still support predictive analysis despite uncertainty or noisy data. This way the surrogates can provide a robust alternative for the simulation-based fitness function, that will not misguide the optimisation process under unexpected or dynamic changes.

RQ3 *How effective are surrogates as alternates for the expensive fitness function?* In RQ2 we evaluate the predictive ability of our surrogates. Yet, even highly accurate predictive models may exhibit lower accuracy than the actual system behaviour, that is modelled via the original simulation-based expensive fitness function. This is due to

the inherent prediction errors (e.g., MAPE) of the predictive models. In this RQ we seek to understand how the surrogate-based fitness functions exploit the uncertainty in the predictions of the metrics of interest Q_E and Q_{RT}, towards guiding the search to near-optimal configurations. Uncertainty can have negative effects by introducing inaccuracies in the optimisation results. It is also possible though to achieve further improvements by introducing uncertainty in the optimisation process, because it can effect to smooth the complex fitness landscape and ease the search towards promising candidates [157]. To assess the effectiveness of the devises surrogates as alternate fitness functions we will compare the quality between Pareto configurations obtained with the expensive and surrogate-based fitness functions.

RQ4 *What is the performance gain by using a light-weight surrogate model instead of the expensive fitness function?*

Our SBSE settings (see Section 5.1.4) require a large number of fitness function evaluations that may pose a hindrance to the feasibility of the optimisation process. To reduce the computational burden of time-consuming, simulation-based fitness functions we introduced surrogate models that offer a mathematical formula to estimate Q_{RT} and Q_E as a function of configurations components i.e., the predictor variables (see the predictors summarised in Table 6.3). However the use of alternate fitness functions introduces a new trade-off; while the initial expensive simulation-based fitness function is time consuming, it accurately models the system behaviour. On the other hand the devised surrogates provide a light-weight fitness function but may introduce inaccuracies since the surrogates approximate the system behaviour using history operation traces. This question will measure and compare the performance gains, with respect to time efficiency, from the use of a surrogate-based fitness function.

RQ5 *Is the surrogate performance generalisable to different genetic algorithms?*

As discussed in Section 2.1.2 two basic ingredients are required before the formulation

of any SBSE problem; a representation and a fitness function definition. Based on the above, any metaheuristic search algorithm may be applied next, to explore the problem solution space. So far we have used the popular NSGA-II algorithm for our experiments. In this question we will repeat our experiments with a different metaheuristic, to confirm the generalisation ability of our surrogate models to different state-of-the-art search algorithms.

Our experiments ran on a cluster of Supermicro Servers, with 2 x Intel Xeon E5420 at 2.50GHz, 2GB DDR2 RAM and 120 GB disk space and Linux Suse OS.

Problem Input

As input candidate for the evaluation study we consider a configuration with the following characteristics; 310 VMs, 340 PMs, 20000 BoT tasks. Its QoS is Q_{RT} = 34 seconds and Q_E = 33 MWhrs (average metrics over 40 CloudSim experiment iterations). The number of VMs and PMs have been randomly selected.

7.1.1 Metrics Used

To answer all our RQs (see Section 7.1) we need to analyse and compare the performance outputs of multi-objective optimisation algorithms. Assessing the performance of a single objective optimisation problems typically requires observations about the best solution found. This approach does not apply in the case of multi-objective optimisation because there are sets of best or "non-dominating" candidate solutions, forming a Pareto front. Each candidate solution in the Pareto front is therefore incomparable to the others because no other has better values for all objectives. To enable quantitative comparison among Pareto sets of candidates we employ two quality indicators for multi-objective optimisers, namely *Hypervolume* (I_H) and *Spread* (Δ) [217].

I_H calculates the volume in the objective space covered by members of a non-dominated set of solutions from an algorithm of interest. For minimisation problems like in this work, a *reference point* defines the upper limit of this volume. The I_H concept is illustrated in Figure 7.1. The *larger* the I_H, the better the algorithm, because the more it captures of the non-dominated solutions space. As the scale of the different objectives in the optimisation problem can be very different (e.g. Q_{RT} may range within a few seconds while Q_E may range within thousands of Watt-hours), the objective values are normalized within the range $[0, 1]$ and the value of 1 is used as a reference point for each objective.

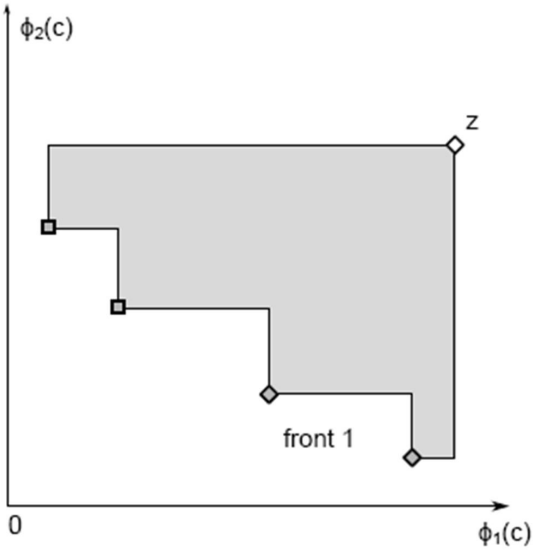


Figure 7.1: Example of hypervolume for a minimisation problem with referenco point z [124].

Δ [190] is a complementary diversity metric that measures how evenly the points in a set of solutions are distributed. *Smaller* values indicate a more uniform spread. A value of zero for this metric would indicate that all members of the Pareto front are equidistantly spaced, though this is rarely the case.

7.1.2 RQ1

In Section 5.1 we have defined the core SBSE artefacts (namely the solution representation, genetic operators and fitness functions) required to reformulate the cloud configuration optimisation problem as a search-based problem. The next step is to apply a meta-heuristic algorithm to explore optimal or near optimal cloud configurations.

One of the most important preconditions for the application of SBSE (see Section 2.1.2), is that the problem candidate has a large and multi-model design space. While in such spaces the application of exact algorithms is practically infeasible due the computational cost, SBSE techniques take advantage of the space characteristics and can help discover acceptable solutions [51]. Otherwise, if the candidate problem for SBSE application has a simpler design space, SBSE might not be the most appropriate choice because it can be slower than an analytic approach due to the repeated trials of the fitness functions to evaluate candidate solutions [89].

According to SBSE practices [89, 8], comparison with random search consists a baseline validation for the efficiency of the customised solution representation, fitness functions and genetic operators as well as a sanity check for the search-based problem formulation. Any SBSE formulation must significantly outperform random search to qualify as worthy of consideration as a successful optimisation technique. Furthermore, in the case of "hard" problems where analytic algorithms are not applicable, the comparison with random search alone suffices to demonstrate that the meta-heuristic technique is capable of producing better solutions for previously uncovered areas of the search-space [89].

To answer this question we use the popular genetic algorithm NSGA-II [57] as implemented in the MOEA framework [1], to explore the space of cloud configurations, encoded as discussed in Section 5.1. Figure 7.2 shows the basic NSGA-II routine.

[1] `http://www.moeaframework.org/`

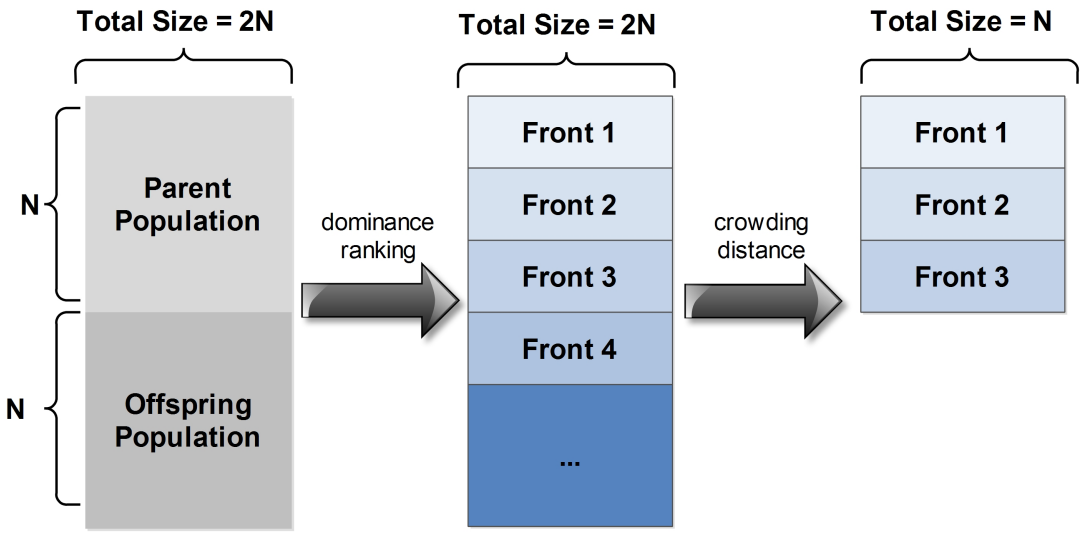


Figure 7.2: The NSGA-II concepts of dominance ranking and crowding distance.

NSGA-II refines the basic GA routine (see Section 1) by applying two additional concepts: (i) *dominance ranking* and (b) *crowding distance*. The generation of our initial population of size 100 (see Section 5.1.1) is followed the breeding phase where mutation and crossover operators are applied to create promising offspring. At this point, after the reproduction has been completed, the new population size is 200 individuals. Each candidate is then mapped to its real-word, phenotype representation using CloudSim. The simulation-based estimations for the metrics Q_{RT}, that denotes the services response time and Q_E, that denotes the cloud energy consumption, consist the fitness score for each individual in the population. Next, the dominance ranking is applied to sort the population into fronts of non-dominated solutions measuring for each individual the number of solutions it dominates, and the number of solutions it is dominated by. The ranking penalises solutions in regions of the objective space which are dominated by many others forcing the exploitation of best candidates. The crowding distance then measures the population density around solutions and aims at maintaining a uniformly spread Pareto set by favouring solutions at less dense regions in every front. The section phase follows, to reduce the population back to

100 by selecting the currently best solution candidates. Our process iterates till 30 generations have been reached.

The bespoke NSGA-II implementation is used as a basis for random search as well, based again on the MOEA framework. The mutation and crossover probabilities are set to zero, eliminating the breeding phase. However, each new generation is instantiated with a new population initialisation, which has the effect that a new population is created at each algorithm iteration. The elite individuals are kept in the population to keep track of the best solutions found to date. Again, the process terminates when 30 generations have been reached. The random search process is summarised in Figure 2.

In comparison to the NSGA-II implementation, random search does not apply genetic operations for searching the candidates space. The comparison between the algorithms' performance will assess whether the genetic operators have been properly designed, to balance the trade-off between exploration of new areas of the search space (via mutations) and exploitation of already known good solution candidates (via crossover). If the use of genetic operators doesn't offer significant performance gains over random search, could mean that (i) the problem is trivial and therefore a cheaper search algorithm as e.g. random search can be used to explore near-optimal configurations or (ii) the current design of genetic operators, the algorithm settings (e.g., number of generations, population size) and/or the fitness functions definition is badly defined.

We run both random search and NSGA-II using the simulation-based expensive fitness function and report the I_H and Δ indicators to compare the quality of solutions achieved by each algorithm. Due to their stochastic nature, evolutionary algorithms may produce different results when applied to the same problem instance. As a randomised algorithm is strongly affected by change, it may find an optimal solution in a very short time or may never converge towards an acceptable solution [8]. It is

Algorithm 2 Random Search

1: **procedure**
2: *Set generation number,* $m := 0$
3: *Define initial population of candidate solutions,* $P(0)$
4: *Evaluate the fitness of each individual* $F_i(P(0))$ *in* $P(0)$
5: **loop**:
6: *Evaluate* $F(m)$
7: *Select* $P(m)$, $P(m) := M(S(m))$
8: $m := m + 1$
9: **exit**: *when stop condition satisfied*
10: **end loop**

hence important to assess the effectiveness of evolutionary algorithms by collecting data from a large enough number of independent runs. To this end, best practice requires the use of inferential statistical testing to robustly assess differences in the performance of the algorithms used [8] (See Section A.2). A *statistical test* assesses whether there is enough empirical evidence to claim a difference between two algorithms. A null hypothesis H_0 is typically defined to initially state that there is no difference between the studied algorithms. A statistical test is then used to verify if H_0 should be rejected.

In our case the H_0 claims that there is no difference between the performance of NSGA-II and random search in the problem of cloud optimisation. Each algorithm is executed 40 times to collect empirical evidence. We then use the non-parametric Wilcoxon test [52] (see also Section A.2.2) to study the probability distribution of the outputs and check for statistical significance. The confidence limit parameter α is set to the widespread used value of 0.05 [8] (see Section A.2.1). According to our results, the H_0 is strongly rejected at the 1% significance level. While Wilcoxon test is safe to use as it makes no assumptions about underlying data distributions, it is inadequate to merely show statistical significance alone; we must also show that the effect size is worthy of interest. To additionally assess the effect size we perform a Cohen effect size test A.2.2 (see also Section A.2.3) in all 80 results. The Wilcoxon

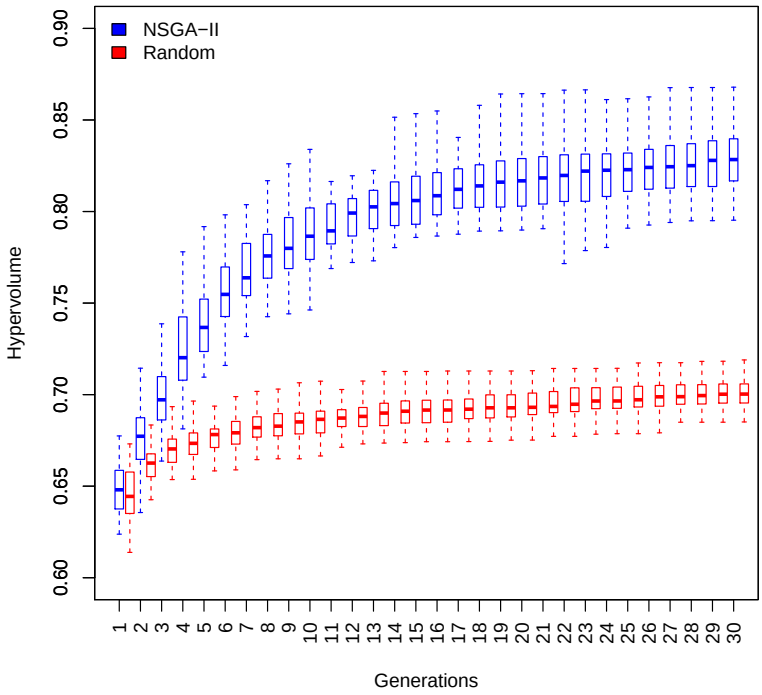


Figure 7.3: Hypervolume evolution through optimisation with NSGA-II and random search.

test results showed that for all 40 experiments the quality indicators achieved by NSGA-II are significantly better than those of random search with a Cohen effect size "high" ($d > 0.8$) . Table 7.1 summarises the mean and standard deviation for the quality metrics achieved by each algorithm. To support qualitatively our point Figure 7.3 presents a visualisation of the hypervolume evolution till each algorithm terminates. We observe that the NSGA-II achieves higher quality solutions at all times. Figure 7.3 also highlights that our choice of 30 generations and population size of 100 are reasonably good settings for the problem, since the NSGA-II hypervolume approaches the maximum value and after generation 20 the hypervolume indicator does not significantly change. Finally Figure 7.4 indicate how much NSGA-II and random search improve the initial population.

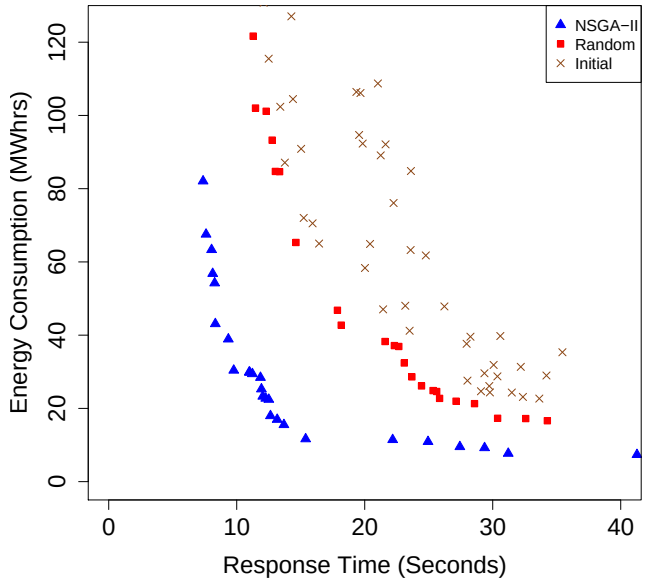


Figure 7.4: NSGA-II and random search Paretos in comparison to the initial population. NSGA-II achieves an improvement up to 79% for response time and 88% for energy consumption w.r.t the problem input.

	$\mathbf{I_H}$		$\mathbf{\Delta}$	
Algorithm	μ	σ	μ	σ
Random	0.7	0.01	0.03	0.01
NSGA-II	0.83	0.02	0.009	0.005

Table 7.1: Quality indicator values of random search and NSGA-II using the expensive fitness function.

7.1.3 RQ2

Our validation results so far have demonstrated that the proposed formulation of cloud optimisation as SBSE is efficient and that indeed the studied problem is not trivial. Our current SBSE settings with a population size of 100 and termination condition of 30 generations require a total of 3000 calls to CloudSim, for estimating the fitness of each individual explored by NSGA-II. As the complexity of the meta-heurisric search highly depends on the complexity of the fitness function [86], our optimisation framework may require from $\sim$ 7 hours to $\sim$ 7 days to complete,

depending on the size of the assigned workload (see Section 5.1.4). Towards reducing the computational burden of time-consuming, simulation-based fitness evaluations we have developed a series of surrogate models as discussed in Chapter 6. The surrogates offer an alternate, analytical expression to model the fitness function, that is orders of magnitude computationally cheaper to use than the initial simulation-based function. The new analytical expression models the relationship between a series of influential variables and the variables of interest Q_E and Q_{RT}. Here the surrogates are functions of the workload size, request size, number of VMs and allocated CPU as Table 6.3 shows. The models have been trained using a dataset of 1000 samples, collected from the CloudSim case study as described in Section 6.2.2.

In Section 6.2.4 we have measured the accuracy of the surrogates by computing widely used error metrics; the Root Mean Squared Error (RMSE), the Mean Absolute Error (MAE) and the Mean Absolute Percentage Error (MAPE). The results are summarised in Table 6.8. RMSE and MAE have the same measuring units as the training dataset, and therefore can only be used for comparison with models of the same measuring units. For example the RMSE and MAE for the metric Q_{RT} is in seconds while for Q_E is in Watt×hours (Whrs). Regarding Q_{RT}, RMSE estimations suggest that MARS and GPs provide more accurate predictions of the services' response times. MAE is not useful here as it is identical for all Q_{RT} models. Regarding Q_E RMSE and MAE unanimously suggest that SVR is the most accurate model.

In comparison to the above, MAPE is expressed in generic percentage terms to indicate how close are the predicted to the actual data. While there is no standardised benchmark to assess MAPE accuracy, Gambi [77] has used a MAPE below 20% to indicate accurate surrogates. In our case the MAPE for all models suggests excellent performance as it ranges within $\sim 5\%-\sim 12\%$. In particular for modelling Q_{RT} MARS scores the best performance with MAPE $\sim 6\%$ and for modelling Q_E SVR

has the best performance with MAPE $\sim 6\%$. Both results agree with the RMSE and MAE findings above.

Our results have shown that the surrogates can efficiently capture the behaviour of a cloud system. The accuracy estimations above reflect scenarios when the cloud datacenter operates within the workload and capability settings of our use case (see Section 3.6 and Table 6.2.2). Up to this point the training data have been preprocessed and collected during controlled experiments.

However, this work aims to apply the surrogates on-line. Therefore we need to further assess how the surrogates perform under uncertainty, noisy data and emerging services behaviours. For example the workload of a web-based system may increase or decrease over time; or it may even change dramatically when events such as Christmas shopping or acquisition of other companies result in a higher number of internal system users [124]. Such unexpected events may not be captured by the initial training dataset. Additionally in real systems the training data are typically collected by monitoring systems, that may introduce noise in the datasets [77].

To model uncertainty in our experiments we extrapolate the parameter settings outside the ranges used to build the surrogates, to examine the robustness of the models in uncontrolled working conditions. More specifically we extrapolate the workload size in two directions; outside the lower and maximum limits used in the training dataset, shown in Table 7.2. The resulting parameter combinations effect in varying levels of workload intensity and variability, shown in Table 7.3.

Predictor	**Interpolation Setup**
Load	[20000 – 90000] BoT jobs

Table 7.2: Interpolation configuration for the predictors.

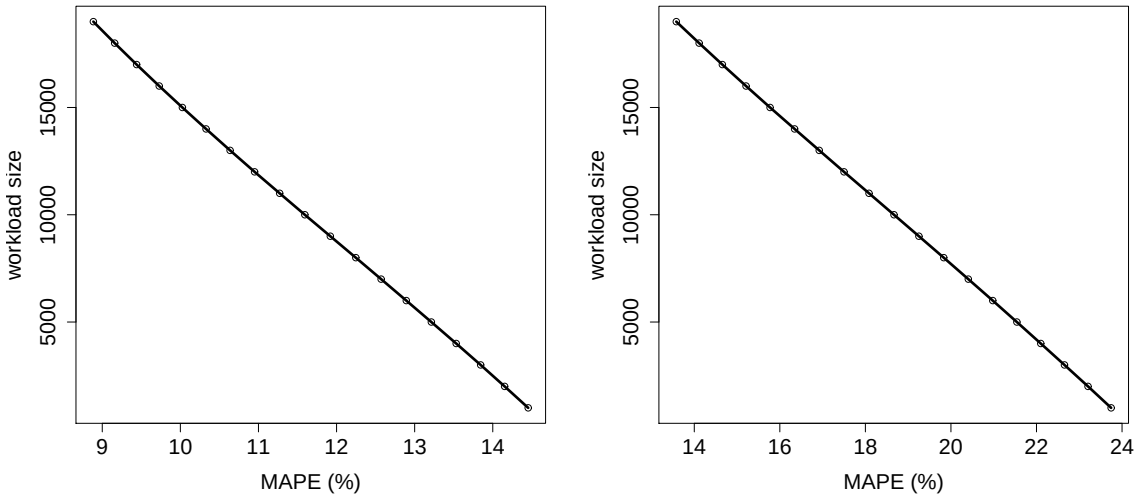
Regarding the prediction of Q_{RT} the MARS and CART surrogates fail to reasonably respond to unexpected workload conditions. The MAPE error for the CART

Predictor	Extrapolation Setup
Load	$[1000 - 19000]$ and $[91000 - 200000]$ BoT jobs

Table 7.3: Extrapolation configuration for the predictors.

models remain static for all extrapolation scenarios at 8.8% while for MARS at 5.9%, equalling the interpolation measurements. The error measures are static because the models output the same prediction value for all inputs. This means that both models do not manage to reasonably respond to unknown workload conditions and to differentiate the predictions between extrapolation and interpolation data. We observe that simplistic regression models do not suffice to model the cloud configurations behaviour.

The results for SVR and GPs surrogates are summarised in Figures 7.5 and 7.6. Compared to the interpolation results in Table 6.2.4 we observe a notable deterioration of the models predictive ability. In general, we expected this outcome because the new data introduce uncertainties and the models have to deal with unexplored regions of the parameter space. The predictive power of the models differs for smaller and larger workload sizes. Scenarios with smaller workloads are predicted well and in particular SVR manages a worst case MAPE of 14% at workload size of 1000 BoTs, which still remains under the indicative 20% accuracy threshold [77]. GPs perform slightly worse with a worst case MAPE of 24% at workload size of 1000 BoTs. There is a higher degradation of predictive ability of the surrogates for larger workload extrapolation scenarios. The worst case MAPE is 30% for SVR and 35% for GPs at workload size of 200000 BoTs. Overall we observe that the size of the new workloads is approximately linearly proportional to the MAPE degradation. Values close to the training limits exhibit satisfactory accuracy that closely resemble the interpolation performance. The more we move outside the training limits though, worse MAPE is obtained. As Figure 7.6 illustrates, our experiment also shows that the deterioration



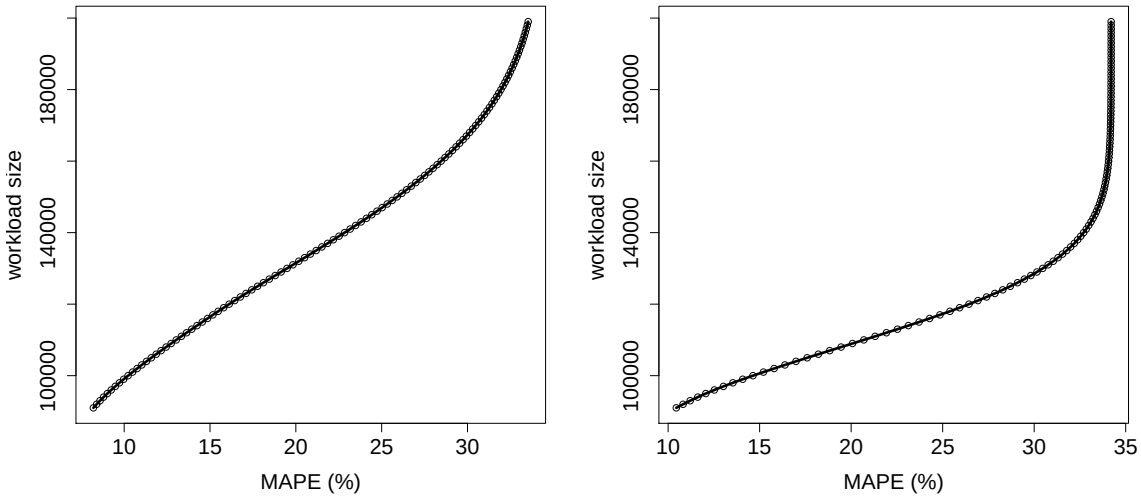
(a) Extrapolation on SVR.

(b) Extrapolation on GPs.

Figure 7.5: Extrapolation for smaller workloads.

on MAPE performance is less aggressive on GPs than SVR surrogates. Interestingly, GPs eventually manage to stabilise the worst case performance at 30%. Overall, the SVR and GPs models show promising results, especially given the fact that the extrapolation values were not only near the training limits where reasonable predictive ability should be expected, but ranged throughout values up to $\sim 120\%$ outside the training data limits. The results are similar for Q_E. To compare, a recent study that builds predictive models for virtualised storage systems [153] extrapolates the parameter values up to $\sim 20\%$ outside the training parameter rages and the corresponding MAPE effect ranges from $\sim 20\%-\sim 119\%$ [153].

To assess the effects of possible increased MAPE errors during the cloud optimisation process, we measure the hypervolumes of the surrogate-based optimisation while the workload range varies outside the training limits. Table 7.4 summarises the achieved hypervolumes in the extrapolation limits using SVR. We observe that when the extrapolated workload values are close to the training limits of minimum 20000 and maximum 90000 the quality of the optimisation output is very robust. In



(a) Extrapolation on SVR.

(b) Extrapolation on GPs.

Figure 7.6: Extrapolation for larger workloads.

particular for workload values 19000 and 91000, the hypervolume indicator measures 0.82 that is approximately equal to the interpolation scenario value of 0.83 (see Table 7.1). The outer extreme extrapolation limit for workload size 200000 achieves a hypervolume value of 0.7 (see Table 7.1). This result suggests that at this extreme point of 30% MAPE the search-based optimisation has been degraded to random search. Interestingly, in the lower extreme extrapolation limit for workload size 1000 we measure a hypervolume value of 0.88 which is higher than the interpolation scenario of 0.83. As we discuss next, in RQ3 (see Section 7.1.4) this improvement can be explained with the *bless of uncertainty* [157] in approximation-based evolutionary computation. The bless of uncertainty highlights the positive impacts by the approximation inaccuracies of the surrogates in the evolutionary search. For briefness we do not display the hypervolume findings for the GPs surrogate, since they are very similar with the Table 7.4, described above.

The ability of surrogates to tolerate unexpected environment conditions goes hand in hand with the requirement to adapt to the new conditions. The models must

Extrapolation Workload	I_H
1000	0.88
19000	0.82
91000	0.82
200000	0.7

Table 7.4: Hypervolumes achieved for extrapolated workloads.

therefore be able to be easily retrained using the new data. The reasons that model adaptability is important is two-fold: (i) on the one side the initial training might have not been optimal and (ii) on the other side clouds are dynamic and an initial training set might not be able to continuously capture their runtime behaviour. Following Gambi's methodology [77] we evaluate the adaptability of surrogates by computing MAPE using two datasets; We divide the original training dataset (see Table 6.2.2) in a *future dataset* and a *current dataset*. The future dataset contains data that appear in time after a surrogate has been trained. The current set denotes the current training set of the surrogates, that grows during the models' lifetime. The size of the current dataset is initially configured at 100 samples. Periodically we retrain the current set, increasing its size by 100 new samples and iteratively evaluate the MAPE of current set. If the error measurement decreases we can conclude that the surrogates are improving. This gives us an intuition of how the training set size relates to the models' predictive ability. We ignore CART and MARS since the previous step shows that they are unable to exploit new data and only output a single static prediction disregarding the input settings. Since the results for SVR and GPs are similar we summarise our findings for SVR surrogate in Figure 7.7.

We clearly observe that the accuracy of the models increases as we introduce more training data samples. However, saving all the monitoring data is not appropriate because it would result to treating old and new data as equally important. This strategy is not necessarily correct in the dynamic cloud context as it might fail to recognise new trends because new data would be smoothed out by old historical

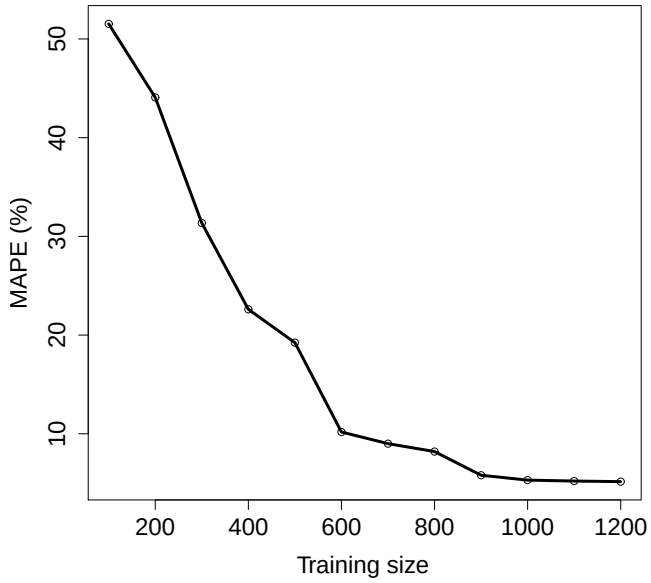


Figure 7.7: Changes in MAPE as the size of current dataset increases.

data. Additionally accumulating data for very similar operating conditions would not provide any advantage to the models [77]. We further observe in Figure 7.7 that a training size above the size of 1000 samples does not considerably improve the model accuracy. We can therefore empirically conclude that the amount of data needed to build a surrogate model with sufficient accuracy is relatively small [77]. This conclusion is in accordance with our discussion in Section 6.2.2 , where we decided a dataset of size 1000 to sample the cloud datacenters parameter space.

In this work we will not go further into sampling training data techniques. However, we assume the presence of a high-lever management framework that filters the monitoring data and maintains only the 1000 most updated samples. For example the *"retain count"* methodology used by Gambi [77], keeps only a number of samples equal to a predefined *retain count* parameter, by replacing old samples with new ones, and retrain the model at the frequency specified by a *retrain period* parameter. Another possible condition to trigger model retraining could be the degradation of error metrics (e.g., MAPE) below a predefined threshold. The training data can

be periodically collected during the normal datacenter operation; each data sample describes a cloud configuration and its corresponding quality (in terms of the metrics Q_E, Q_{RT}).

7.1.4 RQ3

In the case of evolutionary meta-heuristics, the optimization cycle time is proportional to the number of calls to the fitness function [134]. Typically many thousands of fitness function calls are required before a set of near-optimal solutions can be located. The idea of using surrogate approximation models to speed up evolutionary search has recently found its way into SBSE practices. However, surrogate models are not globally accurate and introduce approximation errors. Approximation errors during the search may flaw the optimisation process and introduce false optima since the approximation technique may not be capable of modelling the problem space accurately. The *curse of uncertainty* defines the impairment due to approximation errors of surrogates on the optimisation performance [157]. In this RQ we will assess how the curse of uncertainty affects our surrogates and how close to the original results (as discussed in Section 7.1.2 where the simulation-based fitness function was employed in NSGA-II) is the output of the surrogate-based optimisation.

	$\mathbf{I_H}$		$\mathbf{\Delta}$		
Algorithm	μ	σ	μ	σ	Execution Time
Expensive	0.83	0.02	0.009	0.005	$\sim$ 7 hours
CART	0.51	0.01	n/a	n/a	47 seconds
MARS	0.82	0.02	0.02	0.01	29 seconds
SVR	0.86	0.01	0.02	0.01	65 seconds
GPs	0.87	0.02	0.006	n/a	31 seconds

Table 7.5: Comparison of quality indicators and execution times between expensive and surrogate-based optimisation with NSGA-II.

For each surrogate, we simulate the Pareto solutions of the last generation to compute the true objective space covered using the model. Table 7.5 presents the mean

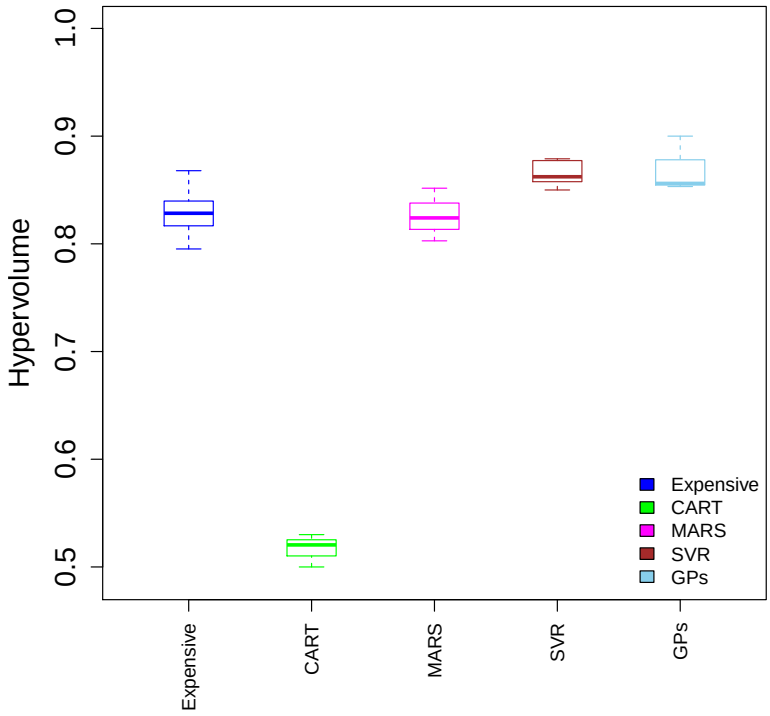


Figure 7.8: Surrogate I_H overview.

values of the quality indicators obtained for 40 runs per algorithm. The values are also qualitatively summarised in Figure 7.8. Regarding I_H MARS is closely approximating the original expensive fitness function, while SVR and GPs manage to outperform it. However, increased Δ values for MARS and SVR suggest a less uniform spread of solutions than the original fitness function. CART has failed to guide the optimisation to good solutions, because it was unable to provide fine-grained differentiation of the quality of neighbouring solutions. We observe that GPs highly benefit from the uncertainty of the approximation error and show excellent performance, outperforming the expensive fitness function in both I_H and Δ. According to Lim et al. [134] approximation errors in surrogates do not always harm. Surrogates are also capable of smoothing multi-modal or noisy landscapes of complex problems. Therefore surrogates may contribute beneficially to the evolutionary search and further improve the original, computationally expensive process. The benefits of using surrogates in the

performance of the evolutionary search is referred to as *bless of uncertainty*. Previous studies by Yao et al. [116] also confirm that smoothed landscape of rugged fitness functions can lead the search to near-optimal solutions easier than the exact fitness model.

At this point it is worth noting that our findings are independent of the starting configuration selection of 310 VMs, 340 PMs and 20000 BoT tasks (see Section 7.1). First, in the population initialisation step (see Section 5.1.1) a population of diverse candidate solutions is generated in order to promote exploration of the search space. The initial population is comprised of random configuration settings (i.e., PM and VM settings) with cardinality ranging from $100 - 1000$ as described in our environment settings description in Section 3.6. The initial population is also seeded with candidates that are possibly closer to a near-optimal solution. The seeding is achieved by mutating the starting configuration applying our six mutation operators (see Section 5.1.2). The invariant among all initial solution candidates is the workload assignment since the generation of diverse configuration settings with the same workload, results in diverse non-functional qualities Q_{RT} and Q_E. Second, in $RQ2$ (see Section 7.1.3) we moved further away from the invariant of workload assignment by extrapolating the workload parameter outside the training ranges and evaluating the performance of the optimisation process. The hypervolume measurements (I_H)on the extrapolated workloads (see Table 7.4) show that the surrogate-assisted optimisation process achieves robust outputs even after extrapolating the workload data $\sim 120\%$ outside the training data limits.

Our results have shown that the choice of approximation techniques affects the performance of the optimisation process. CART was negatively influenced by the curse of uncertainty. MARS achieved satisfying results for I_H while SVR exploited the bless of uncertainty and managed to further improve the originally achieved I_H from 0.83 to 0.86. However, the Δ measurements for both MARS and SVR indi-

cate that the corresponding solutions are not uniformly spread in the problem space. Based on our findings, GPs is the most appropriate surrogate technique for the cloud configuration landscape. The model has benefited from the bless of uncertainty and offers reliable fitness predictions improving both the quality and diversity of the optimisation solutions. In particular GPs has managed to improve the original I_H from 0.83 to 0.87 and the original Δ from 0.009 to 0.006.

7.1.5 RQ4

In this task we assess the gain in performance using light-weight surrogate models by comparing the execution times required to run the evolutionary algorithms on our problem using the expensive fitness function and using each devised surrogate. Our goal is to identify if the models are fast enough to be trained and queried at runtime.

The last column of Table 7.5 summarises the collected execution times to build the corresponding fitness function model and use it to estimate the quality of 3000 cloud configurations, required for one run of NSGA-II. From the results presented in Table 7.5 we conclude that the execution time of the surrogate-based fitness functions is in the order of a few seconds. This is compatible with the control period of online controllers used in Cloud systems, that is also in the order of seconds [77]. According to our data, the proposed surrogates are suitable for on-line usage to speed-up the cloud configurations optimisation process. The execution times of all surrogates are only a fraction of the time needed to run the simulation-based fitness function, that was in the order of hours. Overall the fastest model to use is MARS with 29 seconds, followed closely by GPs with 31 seconds. Combining our findings from RQ3 and RQ4 we conclude that GPs is the most appropriate surrogate as it best balances accuracy and timeliness.

The models, in addition to providing their predictions fast must be also easy to train on-line to support self-adaptiveness, by means of model adaptation. This is

important because as the cloud system evolves and emerging behaviours may appear, the surrogates must be fast to be retrained with new data towards adapting to the new conditions and improving the accuracy of their predictions. If the models are fast to build, they can obtain the most accurate representation of the system without waiting a long time and potentially harm the optimiser's responsiveness, that may compromise performance and violate runtime SLAs. Traditional off-line training approaches are not suitable for cloud systems, as high services availability (i.e., 99.9%) is a critical requirement for the cloud design [188]. The on-demand, self-service nature of service provisioning is also important; consumers can request more or less services, whenever they need, and receive them almost with no waiting time [77].

Table 7.6 explicitly states the build times of our surrogates. Here the models are trained with a sample size of 1000 as described in Section 6.2.2. We observe that our surrogate build times are in the range of seconds; the fastest is MARS with 0.02 seconds while the most time costly is GPs with 2.43 seconds. The considerably increased training time for GPs is expected, since the model constructs correlation matrices among all points in the training data [63], that has a theoretical complexity of $\mathcal{O}(n^2)$ [77] where n is the data size.

Model	**Build Time**
CART	0.03 *seconds*
MARS	0.02 *seconds*
SVR	0.29 *seconds*
GPs	2.43 *seconds*

Table 7.6: Build times of surrogates.

7.1.6 RQ5

We have so far concluded that GPs is the most promising surrogate for use in cloud optimisation. Towards generalising its efficiency, we also evaluate the use of GP surrogates with other state-of-the-art evolutionary algorithms. To this end, we replace

the meta-heuristic NSGA-II with ϵ-MOEA [119] to search for near-optimal configuration candidates. The algorithm ϵ-MOEA has been proposed towards balancing the competing requirements for (i) convergence to the true Pareto-optimal front and (ii) maintaining a well-stributed set of non-dominated solutions in a computationally fast manner. Diversity is maintained by dividing the search space into a number of grids and ensuring that each grid is occupied by a single solution [119]. The algorithm is also *steady state*. Compared with the standard generation approach of NSGA-II where a large portion of the population is selected for reproduction, the steady-state ϵ-MOEA selects only two population individuals to breed. Therefore steady-state algorithms are computationally cheaper. Additionally this strategy is more exploitative than the common generational approach since the parents are maintained in the population for a long time [143].

To introduce fairness in the comparison between meta-heuristics, we use ϵ-MOEA till 3000 evaluations are achieved instead of 30 generations, that is the termination condition for NSGA-II. This is because due to the steady state nature of ϵ-MOEA, when 30 generations are reached, only a minor part of the population has evolved and therefore the quality of the improved candidates is still low.

Table 7.7 summarises the quality indicators gained for using ϵ-MOEA evolutionary algorithm with the simulation-based expensive fitness function versus with GPs-based surrogate fitness function. As before, the surrogate function outperforms the original in both quality metrics. We can therefore conclude that the use of surrogate models can efficiently guide the computational search for to Pareto solutions within seconds independently of the selected metaheuristic algorithm.

Algorithm	$\mathbf{I_H}$ μ	σ	$\mathbf{\Delta}$ μ	σ	Execution Time
Expensive	0.8	0.01	0.02	0.02	$\sim$ 11 hours
GPs	0.89	0.006	0.004	0.002	32 seconds

Table 7.7: Comparison of quality indicators and execution times between expensive and surrogate-based optimisation with ϵ-MOEA.

7.2 Conclusions

In this Chapter we have presented an empirical methodology to evaluate our proposed cloud optimisation framework. Due to the stochastic nature of evolutionary algorithms, statistical testing was used to extract answers to our five research questions.

From the previous discussion we have drawn the following conclusions:

1. In RQ1 we showed that our SBSE formulation outperforms random search, demonstrating that the problem is not trivial and that our approach is effective.

2. In RQ2 we showed that all our surrogates demonstrate robust predictive ability for interpolation scenarios. However, when we use them to predict noisy and unexpected workloads only SVR and GPs demonstrated good predictive ability, concluding that simplistic models as MARS and CART do not suffice to model complex cloud environments. We have explicitly distinguished between extrapolation to small and large workload sizes. Our results showed that the increased MAPE errors may degrade the quality of the output to random search-equivalent in the worst case scenario. In the best case the curse of uncertainty can help mitigate the prediction inaccuracies leading the search to improved quality solutions in comparison with interpolation results. We have therefore concluded that SVR and GPs are robust surrogates that can guarantee acceptable accuracy even when the environment is changing unpredictably.

3. Our results in RQ3 demonstrate that the GPs surrogate can best substitute the original expensive fitness function, since it best balances the metrics of hypervolume (I_H) and diversity (Δ) in the solution outputs. Exploiting the bless of uncertainty a GPs-based fitness function leads to further improvements than the original fitness function in both the quality and diversity of the solutions.

4. In RQ4 we demonstrate that the use of surrogates as fitness functions rapidly speeds up the convergence of the optimisation algorithm. The time to execute the fitness calculations in our optimisation process is reduced from hours to a couple of seconds. We also show that the surrogates are fast to build, and therefore they can be rapidly retrained at runtime when the environment is changing. Overall, our empirical analysis so far showed that the surrogate-based optimisation can tolerate noisy and unexpected data, can provide timely estimations and do not hinder the reactiveness of the framework.

5. Finally in RQ5 we repeat our experiments with a different metaheuristic. In particular we use ϵ-MOEA instead of NSGA-II. Our results show that the improvements of the surrogates in the optimisation process are generalisable to different metaheuristics.

7.3 Threats to Validity

In this section we discuss issues that could pose a threat to validity of our empirical validation.

Using CloudSim Simulator

As the targeted system is an IaaS, a Cloud computing environment that is supposed to create a view of infinite computing resources to users, it is essential to evaluate

the proposed resource allocation algorithms on a large-scale virtualized data center infrastructure. However, it is extremely difficult to conduct repeatable large-scale experiments on a real infrastructure, which is required to evaluate and compare the proposed algorithms. Therefore, to ensure the repeatability of experiments, simulations were chosen as a way to evaluate the performance of the proposed heuristics. In particular the CloudSim toolkit was chosen, as it is a modern simulation framework aimed at cloud computing environments.

While CloudSim is a simulator, it still enables realistic energy-aware cloud configurations' modelling. The available hardware configurations correspond to the real models: HP ProLiant ML110 G4, HP ProLiant ML110 G5 , IBM ×3470, IBM ×3250, IBM ×5670 and IBM ×5675 and the available VMs model the Amazon EC2 instances high CPU medium, extra-large, small and micro. The power consumption of the models is calculated using real data on the corresponding servers power consumption, provided by the results of the SPECpower benchmark [2]. Then the energy consumption of a physical node is calculated as the power consumption over a period of time given by the formula:

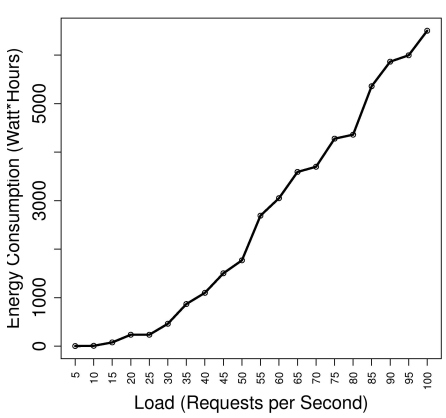
$$E = P \times T \tag{7.1}$$

where P is power and T a time period.

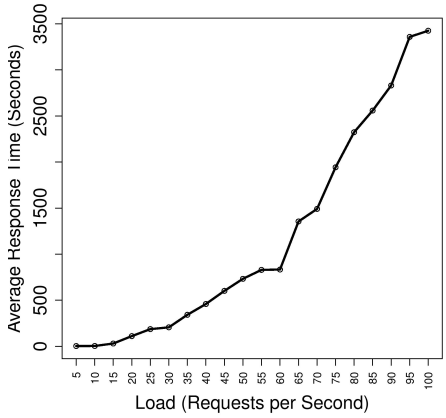
Towards a sanity check of our environment we compare the energy consumption of our simulated servers as the workload increases, with the corresponding SPECpower benchmark results on the real hardware servers.

For this experiment we simulate a cloud of 150 PMs (HP ProLiant ML110 G4 and G5) and 150 random VMs. We generate a web workload, where requests for resources arrive per tier (application, business and database). To stress the system we start with an arrival rate of 5 requests per second and we gradually increase the workload by 5 requests per second, till we reach the arrival rate of 100 requests per second.

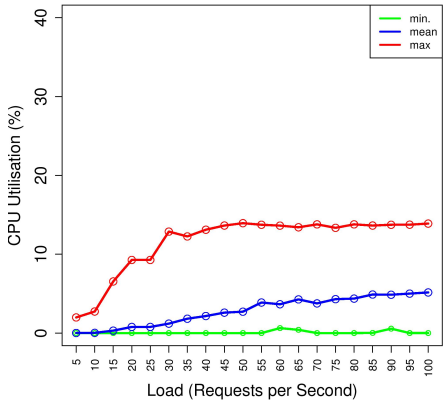
[2] `http://www.spec.org/power_ssj2008/`



(a) Energy Consumption



(b) Response Time

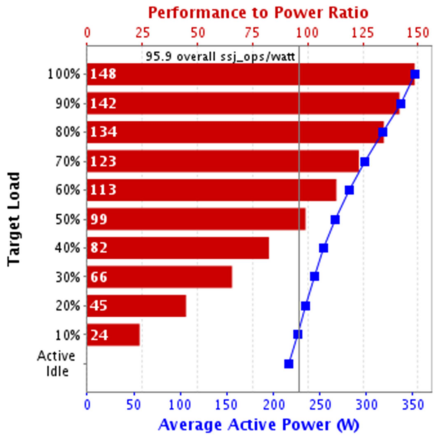


(c) Utilisation Rate

Figure 7.9: Monitored metrics; energy consumption, response times and utilisation ratios[42].

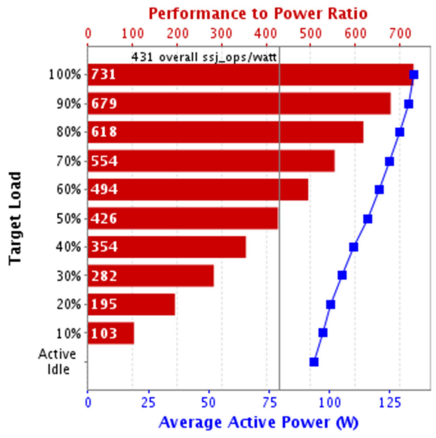
Each distinct arrival rate accounts for one experiment. Each experiment runs for the time period $[t_0, t_{finish}]$ where $t_0 = 0$ and t_{finish} is the time needed for the system to serve the incoming load. We repeat each experiment for 10 times and collect the average values [42].

We observe in Figure 7.9a that in our implementation, the total cloud energy consumption is a linear function with the load size. This is compatible with the benchmarked SPECpower data that have been used to model the server's power consumption. Both power consumption in HP ProLiant ML110 G4 7.10a and in HP ProLiant ML110 G5 7.10b linearly increase with the workload size. We therefore



(a) Power consumption as a function of workload size in (HP ProLiant ML110 G4 [a]

[a] http://www.spec.org/power_ssj2008/results/res2011q1/power_ssj2008-20110127-00341.html



(b) Power consumption as a function of workload size in (HP ProLiant ML110 G5 [a]

[a] http://www.spec.org/power_ssj2008/results/res2011q1/power_ssj2008-20110124-00339.html

Figure 7.10: SPECpower benchmark real data on server power consumption.

conclude that our CloudSim extension complies with realistic, energy aware node configurations. Also, Figure 7.9b shows that the average service response time also increases in proportion with the workload. This is what we should expect since VMs use a static resources (i.e, CPU, RAM, bandwidth) slice to execute the incoming requests. Therefore, as the workload size increases more requests are queued and response times increase. Finally, Figure 7.9c indicates that the average utilisation ratio of the datacenter remains $\sim 20\%$ of its total capacity regardless the workload size. VMs and PMs cannot rebalance their capabilities by default, to scale up or down to the actual demand.

Using Bag-of-Tasks Workloads

Another point that could compromise the validity of our evaluation results is the selection of cloud workload profiles. Since one of the objectives of our approach is to optimise the response times of the cloud services, it is important that the considered

workload profiles reflect real world traces and enterprise behaviours. However, researchers in academia face a severe shortage of data. This is attributed mainly to the risk of reverse engineering trade secrets and the leakage of private user data [46]. As a result of these restrictions we have considered in this work only synthetic workload traces. We have based the selection of synthetic workload models on the analysis of a production cloud trace from Google, that was made available to researchers in 2011 [198]. The data was released to facilitate the understanding of real cloud workloads. The trace describes thousands of *jobs*, each composed of thousand *tasks*. Each task is specified by a set resource requests (i.e., RAM and CPU needed). For each tasks the trace also indicates its submission time and the machine it executed [166].

Reiss et al. [166] have analysed the trace exposing the main identified characteristics to provide insights on modelling realistic cloud workloads. According to the analysis the cloud workload profiles are heterogeneous and span the patterns of scientific computing and high-performance computing. Scientific computing workloads are comprised of large sets of sequential tasks [105] while high-performance workloads consist of batch queueing systems that typically ran CPU-bound programs and often require multiple machines simultaneously for a long period of time [166]. Reiss et al. conclude that the trace tasks are characterised by the parameters of *job duration* and *task shape*. The majority of monitored jobs were short and lasted for only minutes. A small portion of the jobs ($< 1\%$) may last for several hours or even days. Each task is identified as a resource request, which indicate the amount of CPU and memory that the task requires. The requested resources appear to form a *heavy-tailed* distribution. Reiss et al. propose the use of heavy-tailed distributions such as Weibull to approximate the observed behaviours.

Based on the above insights we have resorted to the Bag-of-Tasks (BoT) model [105] (see also Section 3.6) to approximate cloud application workloads. A BoT is used to describe a set of sequential jobs, that share the same executable (i.e., comprise a

single application). Iosup at al. [105] have used over fifteen real grid workloads publicly available via the Grid Workloads Archive [104] to specify and validate the BoT model as representative for the workloads of large-scale distributed systems. The BoT model focuses on three main aspects: (i) the *arrival patterns*, (ii) the*BoT size*, and (iii) the *intra-BoT characteristics*. For each characteristics, the corresponding data from the traces are fitted to low complexity statistical distributions. Then statistical testing is employed to validate the quality of the distribution fit.

The BoT arrival patterns are modelled in two steps: first a Weibull distribution describes the inter-arrival time (IAT) between consecutive BoTs in peak hours and second, an additional Weibull distribution models IAT variations in the daily submission cycle. The BoT batch size varies between a minimum size of $2-4$ to a maximum size of 1000 jobs. A Weibull distribution is selected to model the average size. The intra BoT characteristics are specified by the average task runtime (ART)(i.e., duration) and runtime variability i.e., variation of the runtimes of the tasks belonging to the same BoT. The task runtime is modelled via a Normal distribution and the variability via a Weibull distribution. Finally we use a Pareto distribution to model the heavy-tailed resource requirements of each task within the BoT [15]. The parameters of the distributions are summarised in Table 7.8 below.

BoT characteristic	**distribution**
IAT	$W(4.25, 7.86)$
IAT variations	$W(1.79, 24.16)$
BoT size	$W(1.76, 2.11)$
ART	$N(2.73, 6.1)$
ART variability	$W(2.05, 12.25)$
Resources size	$P(10, 1000)$

Table 7.8: The used parameter values for the BoT characteristics based on [105]. N, P and W stand for the Normal, Pareto and Weibull distributions.

The devised BoT model approximates the cloud workloads as analysed by Reiss et al. from the google trace analysis [166]: The majority of tasks are short-lived, while a small minority takes longer to execute due to the ART variability. Additionally

the resource requests and size of BoTs are heavy tailed since they follow Pareto and Weibull distributions. Based on the above we can conclude that while our workloads are synthetic, we still have injected real world characteristics.

Chapter 8

Conclusions

This thesis addressed the challenge of devising a generic approach to applying surrogate-assisted Search-Based Software Engineering (SBSE) techniques to the challenges found in designing elastic cloud datacenters. Elasticity, a term originating from the fields of physics and economics to denote sensitivity to changes [95], is a property that differentiates clouds from traditional datacenters. In particular elasticity is defined by (i) the precision of adaptation to the environmental changes and by (ii) the speed of adaptation. The first elasticity attribute ensures the system resources are matched to the actual demand as closely as possible to eliminate wastage while the second attribute ensures satisfactory performance under dynamically changing conditions. As a result of an elastic design, business can minimise the risk of uncertainty and achieve continuous services availability at lower costs [108].

So far there has been little effort to provide near-optimal reconfigurations at runtime, towards enabling rapid elasticity in cloud datacenters. Commonly adopted approaches by enterprise clouds that are based on user specific rules are difficult to set-up, do not scale well with the system complexity and may not be effective in the presence of emerging behaviours and unexpected working conditions [77]. More potent research contributions have not yet managed to address the optimality versus

timeliness challenge towards runtime configuration management; very accurate optimisation algorithms need a lot of data and time to converge while more simplistic heuristics are fast to process but provide poor estimations.

To address the issue this thesis investigated SBSE, that has proven to be adept at discovering optimal solutions to a broad range of software engineering problems [90, 6, 12, 68, 51, 81]. To overcome the problem of time-consuming fitness function calls to estimate the candidate solutions' quality we introduce lightweight surrogate models. Surrogates are based on statistical regression techniques to construct an alternate, explicit expression of the fitness function based on data describing the mapping between the configurations' design parameters and the quality of the design. Finally surrogates were integrated to our proposed SBSE problem formulation to explore our initial research hypothesis:

A generic surrogate-assisted search-based reformulation of the cloud configuration improvement problem, would enable the high-quality optimisation results achieved with SBSE to a wealth of existing research, to be applied at runtime.

The proposed framework is suitable to manage on-line adaptation of resources towards maintaining a specified level of QoS in the face of changing workload conditions whiling minimising the cloud operating costs. To address our research hypothesis, the following benchmarks were reached:

- Identification of existing research that applies optimisation techniques to engineer elasticity in cloud computing IaaS configurations, and propose extensions where appropriate.
- Formalisation of an abstract cloud IaaS meta-model to describe the cloud infrastructure structure and degrees-of-freedom, towards engineering an automated improvement process to search for good quality configuration candidates.

- Reformulation of the cloud configuration optimisation problem as SBSE problem through the design and implementation of appropriate (i) solution representation encoding, (ii) genetic operators and (iii) simulation-based fitness-function to evaluate the candidates' QoS.

- Collection of data traces and extraction of influential variables that can be used to explain the QoS metrics of interest; mean response time (Q_{RT}) and energy consumption Q_E.

- Study and utilisation of diverse regression techniques to construct surrogates of different complexity. The surrogates address the lack of an analytical formula to describe the complex relationship between resource allocation, incoming workloads and the end-to-end QoS metrics Q_{RT} and Q_E in cloud IaaS configurations.

- Integration of the current state-of-the-art cloud simulation platform CloudSim [36] the open source evolutionary meta-heuristics platform MOEA [1] and the statistical environment R [2], to create an appropriate environment for the evaluation of the proposed surrogate-assisted SBSE optimisation framework. We also developed realistic synthetic workloads to simulate dynamic workload conditions.

8.1 Thesis Contributions

The contributions made in this thesis are summarised below.

A survey of the literature of cloud datacenter resource provisioning methodologies. Chapter 2 presented a survey of previous work on resource provisioning methodologies. The survey initially investigated resource provisioning techniques for traditional virtualised datacenters and analysed how these methods

[1] `http://www.moeaframework.org/`
[2] `http://www.r-project.org/`

are being updated to suit the new operating model of cloud datacenters. The surveyed literature was classified under two main resource provisioning methods; proactive and reactive. The literature review showcased the limitations of the existing contributions, namely the difficulty to balance the optimality and timeliness trade-off in the configuration optimisation processes. The lack of explicit reporting of convergence times was identified as a limitation towards proving the runtime applicability of proposed solutions.

Demonstration of the limitations of traditionally used methodologies for proactive resource provisioning via an industrial case study. In Chapter 3 we posit that elasticity is the central requirement towards engineering cloud datacenters. We use time-series analysis to develop a proactive methodology that will anticipate demand peaks, considering enterprise workloads from CAS software [3]. Our analysis showed that while proactive mechanisms are important to anticipate changes, predictions at design time are not always accurate. Based on this observation we formalise a vision for elastic dataceneter design, where proactive and reactive methods coexist. To differentiate from existing contributions, we focus on the challenges of the reactive resource adaptation.

Formalisation of the cloud IaaS configurations design space with the specification of a generic cloud meta-model. Chapter 4 introduced a formalisation of the cloud infrastructure core structures and degrees of towards enforcing design constraints and specifying which valid changes may be implemented to tune the configuration quality attributes without affecting the functionality of the system. The specification of the cloud meta-model facilitates the automated improvement process is to find meaningful configuration alternatives, because the architecture

[3] `http://www.cas.de/start.html`

structures do not change arbitrarily during optimisation. To the contrary each configuration that conforms to our meta-model adhere to predefined design constraints, that guarantee meaningful exploration of the design space.

Reformulation of the cloud configuration optimisation problem with Search-based Software Engineering (SBSE). In Chapter 5 we we apply SBSE techniques to encode the challenge of optimising cloud configurations as a search problem. As a result pareto-optimal configuration candidates i.e., candidates that optimally balance many conflicting quality criteria can be tracked. The application of meta-heuristics as part of the SBSE reformulation was identified as a natural fit to the cloud optimisation problem because they (i) treat the problem to be optimised as a black box (ii) are simple to implement as one can make progress with only two ingredients: a choice for representation of the problem and a definition of fitness function (iii) are robust and with appropriate parameter tuning their performance comfortably outperforms purely random search (iv) can demonstrate scalability though parallel execution of candidate fitness computations (v) are any-time algorithms (i.e., may trade deliberation time for quality of results) and therefore are appropriate for time critical domains. Based on the cloud meta-model formalisation, we defined a configuration representation that is amenable to meta-heuristic search and appropriate genetic operators to efficiently explore the design space by balancing exploitation of already known good solutions and exploration of unknown areas of the design space. We finally introduced two simulation-based fitness functions to measure the quality of cloud configuration candidates.

Formulation of surrogate models to approximate the computationally expensive simulation-based fitness functions In Chapter 6 we develop of series of linear and non-linear regression-based models to approximate mathematical for-

mulas that model the complex relationships between cloud configuration resources, workloads and QoS. A representative cloud operating history trace was obtained using simulation. Based on the collected data, the minimum amount of predictor variables was identified to adequately explain the QoS metrics of interest mean response time (Q_{RT}) and energy consumption (Q_E) without incurring unnecessary complexity to the models. The devised surrogates are finally used as alternative fitness functions in our SBSE cloud optimisation problem, to achieve fast convergence times.

Evaluation of the performance of the proposed surrogate-based SBSE framework to optimise cloud configurations at runtime. In Chapter 7 our evaluation analysis first showed that the cloud configurations optimisation problem is non-trivial. Next we demonstrated the feasibility of applying SBSE to the cloud configurations optimisation problem for achieving near-optimal trade-off solutions. We also showed the ability of the devised surrogates to (i) accurately capture the non-functional behaviour of cloud IaaS environments; (ii) to provide precise predictions not only under controlled conditions but also with noise and inaccuracies in the data, that might be introduced in real systems and (iii) to provide estimations in a timely manner. The use of surrogates (i.e., Gaussian Processes) as alternate fitness functions demonstrated that due to the bless of uncertainty, further improvements than the original simulation-based fitness function were achieved. Finally the use of surrogates greatly improved the convergence of our SBSE optimisation framework, reducing the optimisation process execution times from hours to seconds.

A generic optimisation framework. Overall our proposed methodology of combining SBSE and surrogates can be generalisable to many different problem domains since no explicit assumptions are made on the properties of the underlying

research problem. One can make progress by identifying the properties of the problem domain, and thereafter formulating appropriate representation encodings, genetic operators, fitness functions and predictor variables to explain the qualities of interest. For example Efstathiou et al. [65] applied the same methodology to achieve computationally efficient best trade-off service compositions in the context of infrastructureless mobile ad-hoc networks.

8.2 Assumptions and Limitations

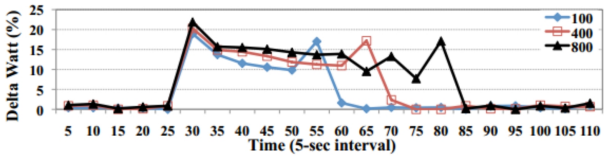
In this study we have assumed that a software cloud monitoring tool exists to provide notifications of changes or SLA violations in the cloud environment. Such notifications will consist triggers to call our optimisation framework and start the improvement process to the existing configuration. Furthermore we have made the working assumption that the scaling of infrastructure does not require additional time and computational expenses. In particular we have assumed that VMs and PMs are able to start or stop operating in near real-time and consistently across infrastructure developments. However, in real scenarios it may take from seconds to a few minutes to have new instances ready [77]. With this assumption, we also note that the overall infrastructure scaling speed can be only as fast as the individual actuators permit; If actuators take too much time to act compared to the workload dynamics, then either the effects of adaptation appear too late and cause performance breaches or more tolerant SLAs will be formed to tolerate the additional complexities. Additionally VM migrations across PMs are also associated with additional latencies and energy consumptions costs, that have not explicitly considered in this thesis. Finally the formulation of Gaussian surrogates, that has been identified as the best performing fitness function approximation, introduces the assumption that the training data-

traces are either noise-free or have Gaussian noise, i.e., noise with zero mean and constant statistical variance for all observations.

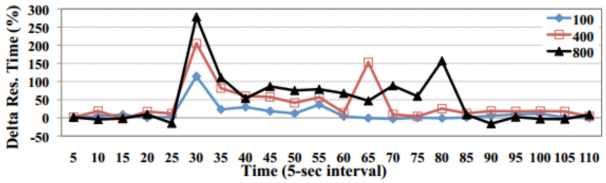
8.3 Open Research

In this section we describe some potential avenues of future work.

- **Modelling of transient adaptation costs:** In this work we have considered the trade-offs between performance, costs and computational complexity, that are inherent in infrastructure optimisation solutions. In addition to the above, infrastructure providers must also consider the trade-offs between adaptation actions and their benefit, since tuning a configuration along its degrees-of-freedom (e.g., by triggering VM migration actions) is not free. While virtualisation technology has made great advances in reducing downtimes during migration to a few milliseconds (e.g., [120]) the end-to-end performance and power consumption impacts can be significant. As Figure 8.1 shows that the increase in power consumption and end-to-end response time depends on the workload type and is incurred over a substantial time period.



(a) Power consumption



(b) Response time

Figure 8.1: Cost of a single VM migration for a three-tier web application. Three deifferent workloads of 100, 400 and 800 concurrent users are shown. [114]

.

Previous research [114] has measured the costs of adaptation actions experimentally offline. In our work we have considered the escalation levels proposed by Maurer et al. [144] towards prioritising cheaper adaptation actions during mutation in the evolutionary optimisation process (see Section 5.1.2). However, the majority of current infrastructure management framework assumes that transient costs are negligible. As a result a formal model of the adaptation costs is required to facilitate the wider study and reasoning of the cost versus benefit trade-off effects.

- **Formalisation of a many-objectives optimisation problem:** In this thesis, as in the majority of cloud optimisation frameworks in the current literature, the authors restrict their efforts to simultaneously balance two to three optimisation objectives. However, in real life there might be a plethora of optimisation requirements, that are possibly in mutual conflict with each other. For example developing high quality cloud software can involve the engineering of a series of quality properties as reliability, modifiability, performance, portability, interoperability security, testability, and usability [19]. In addition economic considerations such as time to market and return of investment are also a major driver for the platform development. Considering more than tree of the aforementioned quality properties in a real world application leads to the formulation of a *many-objective* optimisation problem. This direction of research introduces new challenges for the dynamic infrastructure adaptation problem as many-objective optimisation algorithms tend to incur high computational costs. The large number of objectives leads to further difficulties with respect to computation, visualization, and decision making, that are considered open problems for the multi-objective evolutionary algorithms (MOEAs) community.

- **Combine optimisation at different levels of the cloud stack:** This thesis has studied the optimisation of cloud infrastructure, abstracting away the inner details of cloud services and their components, that comprise SaaS i.e., the highest level of the cloud hierarchy. The SaaS layer involves migrating and deploying enterprise software to the cloud infrastructure, that still entails a wealth of challenges and potential pitfalls to stakeholders. For example it is challenging to select the best VMs allocation to host an enterprise component-based application since the design space that spans for all possible deployment options is huge, the elements of a single deployment option exhibit complex non-linear interdependencies while deployment option evaluations are very time-consuming and can take from a few minutes to several hours [74]. Similar challenges appear in the problem of cloud services composition where complex applications are created by aggregating services to provide composite functionalities that none of the individual services could provide by itself. To fully exploit cloud elasticity optimisation cannot only be restricted to the challenge of mapping VMs to hardware servers. The challenges of software migration and services selection and composition should be also studied in parallel to provide a robust solution.

Appendix A

Appendix

A.1 Covariance and Correlation

Assuming a set of data observations on a response variable Y and a predictor variable X, *covariance* $Cov(Y, X)$ measures the direction while *correlation* $Cor(Y, X)$ measures the strength of the relation between variables X and Y. $Cov(Y, X)$ is defined as:

$$Cov(Y, X) = \frac{\sum_{i=1}^{n}(y_i - \bar{y})(x_i - \bar{x})}{n - 1} \tag{A.1}$$

where n the total number of data observation subjects, y_i is the observation $i \in [i, n]$ for the response Y, x_i is is the observation $i \in [i, n]$ for the predictor, $\bar{x}$ is the mean of the predictor and $\bar{y}$ is the mean of the response. If $Cov(Y, X) > 0$ there is a positive relationship between the variables but if $Cov(Y, X) < 0$ the relationship is negative. $Cor(Y, X)$ is defined as the ratio of the covariance to the standard deviations of the two variables:

$$Cor(Y, X) = \frac{Cov(Y, X)}{s_y s_x} \tag{A.2}$$

where $s_x = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}}$ and $s_y = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \bar{y})^2}{n-1}}$ denote the standard deviation of X and Y accordingly.

A.2 Statistical Testing

To enable statistical analysis, the studied algorithms should run a large number of times (n) in an independent way to collect information on the probability distribution of each algorithm. A commonly used value for n is at least 30 for each algorithm. A statistical test is then used to verify whether the null hypothesis H_0 should be rejected. The two most used statistical tests are the $t-$test and the Wilcoxon test. A t-test is parametric i.e., makes assumptions on the distribution of the data and aims to compare the mean values of two distributions. The Wilcoxon is non-parametric and therefore raises the bar for the significance of the findings by making no assumptions about the underlying data distributions.

A.2.1 p-Values

There are two possible types of errors when performing statistical testing : (I) The null hypothesis is rejected when it is true and (II) we accept the null hypothesis when it is false. The *p-value* of a statistical test denotes the probability of type I error. The *significant level* α of a test is the highest p-value we accept for rejecting the null hypothesis.

A.2.2 The Wilcoxon Test

The wilcoxon test can be used to assess whether two sets of data show a significant overall difference in the magnitude of a metric of interest. The test assumes that the investigated data groups have been randomly collected and that there is mutual independence between the groups. Under the null hypothesis the samples we are comparing come from populations from the same distribution. If we combine the samples and then assign ranks to each observation allocating the rank 1 to the smallest, rank 2 to the next and so on we shall expect under the null hypothesis that the scores from

the samples would be spread randomly in the rank ordering. If on the other hand there is a significant difference between samples we would observe the one sample to contribute more towards the upper part of the rankings list and the other mainly to the lower part. The wilcoxon test calculates the probability of finding any given difference between the samples under the null hypothesis.

A.2.3 Cohen Effect Size Test

Though it is important to assess whether an algorithm performs statistically better than another, it is in addition crucial to assess the magnitude of the improvement. To analyse such a property *effect size* measures are introduced. Cohen defined d as the difference between the means M_1 and M_2 of the compared data groups, divided by the standard deviation of either group. The effect sizes are defined as small for $d = 0.2$, medium for $d = 0.5$ and large for $d \geq 0.8$.

Bibliography

[1] *Validation in model-driven engineering: testing model transformations*, 2004.

[2] OMG, UML superstructure, version 2.1.1. OMG document formal , 2007.

[3] Virtualization and Multicore Innovations Disrupt the Worldwide Server Market, 2007.

[4] The R Project for Statistical Computing., http://www.r-project.org/.

[5] S. Abdelwahed, Nagarajan Kandasamy, and Sandeep Neema. Online control for self-management in computing systems. In *Real-Time and Embedded Technology and Applications Symposium, 2004. Proceedings. RTAS 2004. 10th IEEE*, 2004.

[6] Aldeida Aleti, Barbora Buhnova, Lars Grunske, Anne Koziolek, and Indika Meedeniya. Software Architecture Optimization Methods: A Systematic Literature Review. *IEEE Transactions on Software Engineering*, 2012.

[7] Kurt Hornik Alexandros Karatzoglou, Alex Smola. *Kernel-based Machine Learning Lab*, 2013.

[8] Andrea Arcuri and Lionel Briand. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 1–10, New York, NY, USA, 2011. ACM.

[9] Danilo Ardagna, GiovanniPaolo Gibilisco, Michele Ciavotta, and Alexander Lavrentev. A Multi-model Optimization Framework for the Model Driven Design of Cloud Applications. In *Search-Based Software Engineering*. Springer International Publishing, 2014.

[10] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Commun. ACM*, 2010.

[11] A. Babyak. What You See May Not Be What You Get: A Brief, Nontechnical Introduction to Overfitting in Regression-Type Models. 2004.

[12] Anthony J. Bagnall, Victor J. Rayward-Smith, and Ian Whittley. The next release problem. *Information & Software Technology*, 2001.

[13] Michelle Bailey. Infrastructure Convergence: The Integration of Technology to Lower Cost and Improve Business Response. *HP*, 2009.

[14] Michelle Bailey. Infrastructure Convergence: The Integration of Technology to Lower Cost and Improve Business Response. Technical report, HP, 2009.

[15] Paul Barford and Mark Crovella. Generating representative Web workloads for network and server performance evaluation. In *Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, SIGMETRICS '98/PERFORMANCE '98, pages 151–160, New York, NY, USA, 1998. ACM.

[16] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, 2003.

[17] Enda Barrett, Enda Howley, and Jim Duggan. Applying reinforcement learning towards automating resource allocation and application scalability in the cloud. *Concurrency and Computation: Practice and Experience*, 2013.

[18] Salman A. Baset. Cloud SLAs: Present and Future. *SIGOPS Oper. Syst. Rev.*, 2012.

[19] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice (2nd Edition)*. Addison-Wesley Professional, 2003.

[20] Steffen Becker, Heiko Koziolek, and Ralf Reussner. The Palladio component model for model-driven performance prediction. *Journal of Systems and Software*, 2009.

[21] Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing. *Future Generation Computer Systems*, 2012.

[22] Anton Beloglazov and Rajkumar Buyya. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in Cloud data centers. *Concurrency and Computation: Practice and Experience*, 2012.

[23] Fabrcio Benevenuto, Csar Fernandes, Matheus Santos, Virglio Almeida, Jussara Almeida, G.(John) Janakiraman, and JosRenato Santos. Performance Models for Virtualized Applications. In *Frontiers of High Performance Computing and Networking ISPA 2006 Workshops*. 2006.

[24] Amip Shahb Graeme Maidmenta Beth Whiteheada, Deborah Andrewsa. Assessing the environmental impact of data centres part 1: Background, energy use and metrics. 2014.

[25] K. Bilal, S.U. Khan, and A.Y. Zomaya. Green Data Center Networks: Challenges and Opportunities. In *Frontiers of Information Technology (FIT), 2013 11th International Conference on*, 2013.

[26] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 2003.

[27] Norman Bobroff, Andrzej Kochut, and Kirk Beaty. Dynamic Placement of Virtual Machines for Managing SLA Violations. In *Integrated Network Management*, 2007.

[28] George Edward Pelham Box and Gwilym Jenkins. *Time Series Analysis, Forecasting and Control*. Holden-Day, Incorporated, 1990.

[29] Robert Bradford, Evangelos Kotsovinos, Anja Feldmann, and Harald Schiöberg. Live wide-area migration of virtual machines including local persistent state. In *Proceedings of the 3rd international conference on Virtual execution environments*, VEE '07, pages 169–179, New York, NY, USA, 2007. ACM.

[30] Jrgen Branke. Evolutionary Approaches to Dynamic Optimization Problems - A Survey -. pages 134–137, 1999.

[31] David Breitgand, Gilad Kutiel, and Danny Raz. Cost-aware live migration of services in the cloud. In *Proceedings of the 3rd Annual Haifa Experimental Systems Conference*, SYSTOR '10, pages 11:1–11:1, New York, NY, USA, 2010. ACM.

[32] Fabian Brosig, Nikolaus Huber, and Samuel Kounev. Modeling Parameter and Context Dependencies in Online Architecture-level Performance Models. In *Proceedings of the 15th ACM SIGSOFT Symposium on Component Based Software Engineering*, 2012.

[33] Rajkumar Buyya, Anton Beloglazov, and Jemal Abawajy. Energy-efficient management of data center resources for Cloud computing: A vision, architectural elements, and open challenges. In *Proceedings of the 2010 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2010)*, 2010.

[34] Rajkumar Buyya and Manzur Murshed. GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing. *Concurrency and Computation: Practice and Experience*, (13-15), 2002.

[35] Rajkumar Buyya, Rajiv Ranjan, and Rodrigo N. Calheiros. InterCloud: Utility-oriented Federation of Cloud Computing Environments for Scaling of Application Services. In *Proceedings of the 10th International Conference on Algorithms and Architectures for Parallel Processing - Volume Part I*, 2010.

[36] Rodrigo N. Calheiros, Rajiv Ranjan, César A. F. De Rose, and Rajkumar Buyya. CloudSim: A Novel Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services. *CoRR*, 2009.

[37] Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, and Maria Luisa Villani. An Approach for QoS-aware Service Composition Based on Genetic Algorithms. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*, 2005.

[38] Emiliano Casalicchio, Daniel A. Menascé, and Arwa Aldhalaan. Autonomic Resource Provisioning in Cloud Systems with Availability Goals. In *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*, 2013.

[39] Sivadon Chaisiri and Dusit Niyato. Optimal virtual machine placement across multiple cloud providers. *2009 IEEE AsiaPacific Services Computing Conference APSCC*, 2009.

[40] Gregory Ganger Randy Katz Michael Kozuch Charles Reiss, Alexey Tumanov. Towards understanding heterogeneous clouds at scale: Google trace analysis. 2013.

[41] Samprit Chatterjee, Ali S. Hadi, and Bertram Price. *Regression Analysis by Example*. Wiley-Interscience, 1999.

[42] Kleopatra Chatziprimou, Kevin Lano, and Steffen Zschaler. Runtime Infrastructure Optimisation in Cloud IaaS Structures. In *CloudCom*, pages 687–692, 2013.

[43] Kleopatra Chatziprimou, Kevin Lano, and Steffen Zschaler. Towards a Metamodel of the Cloud Computing Resource Landscape. In *MODELSWARD*, 2013.

[44] Kleopatra Chatziprimou, Kevin Lano, and Steffen Zschaler. *Surrogate-Assisted Online Optimisation of Cloud IaaS Configurations*. Institute of Electrical and Electronics Engineers (IEEE), 2014.

[45] Kuan-Yu Chen and Cheng-Hua Wang. Support vector regression with genetic algorithms in forecasting tourism demand . *Tourism Management*, 2007.

[46] Yanpei Chen, Archana Sulochana Ganapathi, Rean Griffith, and Randy H. Katz. Analysis and Lessons from a Publicly Available Google Cluster Trace. Technical report, University of California, Berkeley, 2010.

[47] Betty H. Cheng, Rogério Lemos, Holger Giese, Paola Inverardi, Jeff Magee, Jesper Andersson, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Giovanna Marzo Serugendo, Schahram Dustdar, Anthony Finkelstein, Cristina Gacek, Kurt Geihs, Vincenzo Grassi, Gabor Karsai, Holger M. Kienle, Jeff Kramer, Marin Litoiu, Sam Malek, Raffaela Mirandola, Hausi A. Müller, Sooyong Park, Mary Shaw, Matthias Tichy, Massimo Tivoli, Danny Weyns, and Jon Whittle. Software Engineering for Self-Adaptive Systems. Springer-Verlag, 2009.

[48] Leslie Cheung, Leana Golubchik, and Fei Sha. A Study of Web Services Performance Prediction: A Client's Perspective. In *MASCOTS*, 2011.

[49] Ching-Wu Chu and Guoqiang Peter Zhang. A comparative study of linear and nonlinear models for aggregate retail sales forecasting . 2003.

[50] Ching Chuen, Teck Mark, Dusit Niyato, and Tham Chen-khong. Evolutionary Optimal Virtual Machine Placement and Demand Forecaster for Cloud Computing. *2011 IEEE International Conference on Advanced Information Networking and Applications*, 2011.

[51] J. Clarke, J. J. Dolado, M. Harman, R. Hierons, B. Jones, M. Lumkin, B. Mitchell, S. Mancoridis, K. Rees, M. Roper, and M. Shepperd. Reformulating software engineering as a search problem. *Software, IEE Proceedings* -, 2003.

[52] Jacob Cohen. *Statistical Power Analysis for the Behavioral Sciences (2nd Edition)*. Routledge, 1988.

[53] Johan Tordsson David Breitgand, Alessandro Maraschini. Policy-Driven Service Placement Optimization in Federated Clouds. Technical report, IBM Research, 2011.

[54] Kurt Hornik Friedrich Leisch Chih-Chung Chang Chih-Chung Chang David Meyer, Evgenia Dimitriadou. *Misc Functions of the Department of Statistics (e1071)*. TU Wien, 2013.

[55] Claus de Castro Aranha and Hitoshi Iba. The Memetic Tree-based Genetic Algorithm and its application to Portfolio Optimization. *Memetic Computing*, 2009.

[56] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Commun. ACM*, 51(1), January 2008.

[57] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A Fast Elitist Multi-Objective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 2000.

[58] Peter A. Dinda and David R. O'Hallaron. Host Load Prediction Using Linear Models, 2000.

[59] Els Ducheyne, Bernard De Baets, and Robert De Wulf. Is Fitness Inheritance Useful for Real-world Applications? In *Proceedings of the 2Nd International Conference on Evolutionary Multi-criterion Optimization*, EMO'03, pages 31–42, Berlin, Heidelberg, 2003. Springer-Verlag.

[60] C.L. Dumitrescu and I. Foster. GangSim: a simulator for grid scheduling studies. In *Cluster Computing and the Grid*, 2005.

[61] Xavier Dutreilh, Sergey Kirgizov, Olga Melekhova, Jacques Malenfant, Nicolas Rivierre, and Isis Truck. Using Reinforcement Learning for Autonomic Resource Allocation in Clouds: Towards a Fully Automated Workflow. In *ICAS 2011*, 2011.

[62] Xavier Dutreilh, Aurelien Moreau, Jacques Malenfant, Nicolas Rivierre, and Isis Truck. From Data Center Resource Allocation to Control Theory and Back. *2013 IEEE Sixth International Conference on Cloud Computing*, 2010.

[63] M. Ebden. Gaussian Processes for Regression: A Quick Introduction. Technical report.

[64] Dionysios Efstathiou, Peter McBurney, Steffen Zschaler, and Johann Bourcier. Efficiently Approximating the QoS of Composite Services in Mobile Ad-Hoc Networks.

[65] Dionysios Efstathiou, Peter McBurney, Steffen Zschaler, and Johann Bourcier. Surrogate-Assisted Optimisation of Composite Applications in Mobile Ad hoc Networks. In *Proceedings of the 16th Annual Genetic and Evolutionary Computation Conference*, 2014.

[66] Eugen Feller, Louis Rilling, and Christine Morin. Energy-Aware Ant Colony Based Workload Placement in Clouds. In *The 12th IEEE/ACM International Conference on Grid Computing (GRID-2011)*, Lyon, France, 2011.

[67] Eugen Feller, Cyril Rohr, David Margery, and Christine Morin. Energy Management in IaaS Clouds: A Holistic Approach. In *IEEE CLOUD*, 2012.

[68] Chi-Ming Lin Feng Wena. Multistage Human Resource Allocation for Software Development by Multiobjective Genetic Algorithm. *The Open Applied Mathematics Journal*, 2008.

[69] Hector Fernandez, Guillaume Pierre, and Thilo Kielmann. Autoscaling Web Applications in Heterogeneous Cloud Infrastructures. In *IEEE International Conference on Cloud Engineering*, 2014.

[70] Tiago C Ferreto, Marco A S Netto, Rodrigo N Calheiros, and Csar A F De Rose. Server consolidation with migration control for virtualized data centers. *Future Generation Computer Systems*, 2011.

[71] Scott Fortmann-Roe. Understanding the Bias-Variance Tradeoff, 2012.

[72] Ian Foster, Carl Kesselman, and Steven Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *Int. J. High Perform. Comput. Appl.*, 2001.

[73] John Fox. Regression Diagnostics, 2009.

[74] Sören Frey, Florian Fittkau, and Wilhelm Hasselbring. Search-based Genetic Optimization for Deployment and Reconfiguration of Software in the Cloud. In *Proceedings of the 2013 International Conference on Software Engineering*, 2013.

[75] A. Gambi and G. Toffetti. Modeling Cloud performance with Kriging. In *Software Engineering (ICSE), 2012 34th International Conference on*, 2012.

[76] A. Gambi, G. Toffetti, C. Pautasso, and M. Pezze. Kriging Controllers for Cloud Applications. *Internet Computing, IEEE*, 2013.

[77] Alession Gambi. *Kriging-based Self-Adaptive Controllers for the Cloud*. PhD thesis, 2013.

[78] Antnio Gaspar-Cunha and Armando Vieira. A Multi-Objective Evolutionary Algorithm Using Neural Networks to Approximate Fitness Evaluations. *Int. J. Comput. Syst. Signal*, (1), 2005.

[79] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, pages 29–43, New York, NY, USA, 2003. ACM.

[80] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., 1989.

[81] David E. Goldberg. *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*. Kluwer Academic Publishers, 2002.

[82] David E. Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of Genetic Algorithms*, 1991.

[83] Laura Grit, David Irwin, Aydan Yumerefendi, and Jeff Chase. Virtual Machine Hosting for Networked Clusters: Building the Foundations for "Autonomic" Orchestration. In *Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing*, 2006.

[84] Yanfei Guo, P. Lama, Jia Rao, and Xiaobo Zhou. V-Cache: Towards Flexible Resource Provisioning for Multi-tier Applications in IaaS Clouds. In *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, 2013.

[85] James Hamilton. The Cost of Latency, 2009.

[86] M. Harman and J. Clark. Metrics are fitness functions too. In *Software Metrics, 2004. Proceedings. 10th International Symposium on*, 2004.

[87] M. Harman, K. Lakhotia, J. Singer, D. White, and Y. Shin. Cloud engineering is search based software engineering too. *Journal of Systems and Software*, 2013.

[88] Mark Harman. The Current State and Future of Search Based Software Engineering. In *2007 Future of Software Engineering*, 2007.

[89] Mark Harman and Bryan F. Jones. Search-Based Software Engineering. *Information and Software Technology*, 2001.

[90] Mark Harman, S. Afshin Mansouri, and Yuanyuan Zhang. Search-based Software Engineering: Trends, Techniques and Applications. *ACM Comput. Surv.*, 2012.

[91] Mark Harman, Phil McMinn, Jerffeson Teixeira de Souza, and Shin Yoo. Search-Based Software Engineering: Techniques, Taxonomy, Tutorial. In *Empirical Software Engineering and Verification*. Springer, 2011.

[92] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning*. Springer, 2003.

[93] Randy L. Haupt and Sue Ellen Haupt. *Practical Genetic Algorithms*. John Wiley & Sons, Inc., 1998.

[94] Nikolas Roman Herbst, Nikolaus Huber, Samuel Kounev, and Erich Amrehn. Self-Adaptive Workload Classification and Forecasting for Proactive Resource Provisioning. In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, 2013.

[95] Nikolas Roman Herbst, Samuel Kounev, and Ralf Reussner. Elasticity in Cloud Computing: What It Is, and What It Is Not. In *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)*, 2013.

[96] Fabien Hermenier, Xavier Lorca, Jean-Marc Menaud, Gilles Muller, and Julia Lawall. Entropy: a consolidation manager for clusters. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, 2009.

[97] https://www.dropbox.com/.

[98] http://www.rackspace.co.uk/cloud/files.

[99] Nikolaus Huber, Fabian Brosig, and Samuel Kounev. Model-based self-adaptive resource allocation in virtualized environments. In *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '11, pages 90–99, New York, NY, USA, 2011. ACM.

[100] Nikolaus Huber, Fabian Brosig, and Samuel Kounev. Modeling Dynamic Virtualized Resource Landscapes. In *Proceedings of the 8th ACM SIGSOFT International Conference on the Quality of Software Architectures (QoSA 2012)*, June 2012. Acceptance Rate (Full Paper): 25.6%.

[101] Rob J. Hyndman and Yeasmin Khandakar. Automatic Time Series Forecasting: The forecast Package for R. *Journal of Statistical Software*, 2008.

[102] Google Inc.

[103] Google Inc. Google apps engine.

[104] Alexandru Iosup, Hui Li, Mathieu Jan, Shanny Anoep, Catalin Dumitrescu, Lex Wolters, and Dick H. J. Epema. The Grid Workloads Archive. *Future Gener. Comput. Syst.*, 2008.

[105] Alexandru Iosup, Ozan Sonmez, Shanny Anoep, and Dick Epema. The Performance of Bags-of-tasks in Large-scale Distributed Systems. 2008.

[106] David Irwin, Jeffrey Chase, Laura Grit, Aydan Yumerefendi, David Becker, and Kenneth G. Yocum. Sharing networked resources with brokered leases. In *Proceedings of the annual conference on USENIX '06 Annual Technical Conference*, 2006.

[107] Sadeka Islam, Jacky Keung, Kevin Lee, and Anna Liu. Empirical Prediction Models for Adaptive Resource Provisioning in the Cloud. 2012.

[108] Sadeka Islam, Kevin Lee, Alan Fekete, and Anna Liu. How a Consumer Can Measure Elasticity for Cloud Platforms. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, 2012.

[109] Glenn D. Israel. Determining Sample Size, 2013.

[110] David M. Chess Jeffrey O. Kephart. The vision of autonomic computing. Jan 2003.

[111] Y. Jin. A Comprehensive Survey of Fitness Approximation in Evolutionary Computation. *Soft Comput.*, 2005.

[112] Yaochu Jin. Surrogate-assisted evolutionary computation: Recent advances and future challenges. *Swarm and Evolutionary Computation*, 2011.

[113] Page Josyula, Orr. *Cloud Computing: Automating the Virtualized Data Center*. 2012.

[114] Gueyoung Jung, Matti A Hiltunen, Kaustubh R Joshi, Richard D Schlichting, and Calton Pu. Mistral: Dynamically Managing Power, Performance, and Adaptation Cost in Cloud Infrastructures. *2010 IEEE 30th International Conference on Distributed Computing Systems*.

[115] Gueyoung Jung and Hyunjoo Kim. Optimal Time-Cost Tradeoff of Parallel Service Workflow in Federated Heterogeneous Clouds. In *Web Services (ICWS), 2013 IEEE 20th International Conference on*, 2013.

[116] X. Yao K.-H. Liang and C. Newton. Combining landscape approximation and local search in global optimization. 1999.

[117] Evangelia Kalyvianaki. *Resource provisioning for virtualized server applications*. PhD thesis, University of Cambridge, 2009.

[118] Krishna Kant. Data center evolution. *Comput. Netw.*, 53(17):2939–2965, December 2009.

[119] S.Mishra K.Deb, M.Mohan. A fast multi-objective evolutionary algorithm for finding well-spread pareto-optimal solutions.

[120] Christopher Clark Keir, Christopher Clark, Keir Fraser, Steven H, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live Migration of Virtual Machines. In *In Proceedings of the 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation*, 2005.

[121] G. Khanna, K. Beaty, G. Kar, and A. Kochut. Application Performance Management in Virtualized Server Environments. In *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP*, 2006.

[122] Samuel Kounev, Fabian Brosig, Nikolaus Huber, and Ralf Reussner. Towards Self-Aware Performance and Resource Management in Modern Service-Oriented Systems. In *IEEE SCC*, 2010.

[123] Samuel Kounev, Philipp Reinecke, Fabian Brosig, Jeremy T. Bradley, Kaustubh Joshi, Vlastimil Babka, Stephen T. Gilmore, and Anton Stefanek. Providing Dependability and Resilience in the Cloud: Challenges and Opportunities. Springer Verlag, June 2012.

[124] Anne Koziolek. *Automated Improvement of Software Architecture Models for Performance and Other Quality Attributes*. PhD thesis, KIT, Karlsruhe, Germany, 2011.

[125] S. Kundu, R. Rangaswami, K. Dutta, and M. Zhao. Application performance modeling in a virtualized environment. In *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, pages 1–10. IEEE, January 2010.

[126] Dara Kusic, Jeffrey O Kephart, James E Hanson, Nagarajan Kandasamy, and Guofei Jiang. Power and performance management of virtualized computing environments via lookahead control. *Cluster Computing*, 2008.

[127] Eric-Wubbo Lameijer, Thomas Bäck, Joost N. Kok, and Ad P. Ijzerman. Evolutionary Algorithms in Drug Design. 2005.

[128] John OLoughlin Anuz Pratap Singh Tomar Lee Gillam, Bin Li. Fair Benchmarking for Cloud Computing systems. 2013.

[129] Charles Lefurgy, Xiaorui Wang, and Malcolm Ware. Server-level power control. In *In Proceedings of the IEEE International Conference on Autonomic Computing (ICAC*, 2007.

[130] Alexander Lenk, Markus Klems, Jens Nimis, Stefan Tai, and Thomas Sandholm. What's inside the Cloud? An architectural map of the Cloud landscape. In *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, 2009.

[131] Frank Leymann. Cloud Computing: The Next Revolution in IT. 2009.

[132] Bo Li, Jianxin Li, Jinpeng Huai, Tianyu Wo, Qin Li, and Liang Zhong. EnaCloud: An Energy-Saving Application Live Placement Approach for Cloud Computing Environments. In *Proceedings of the 2009 IEEE International Conference on Cloud Computing*, 2009.

[133] Kun Li, Gaochao Xu, Guangyu Zhao, Yushuang Dong, and Dan Wang. Cloud Task Scheduling Based on Load Balancing Ant Colony Optimization. In *Proceedings of the 2011 Sixth Annual ChinaGrid Conference*, 2011.

[134] Dudy Lim, Yaochu Jin, Yew-Soon Ong, and Bernhard Sendhoff. Generalizing Surrogate-assisted Evolutionary Computation.

[135] Mingfeng Lin, Henry C. Lucas, and Galit Shmueli. Research CommentaryToo Big to Fail: Large Samples and the p-Value Problem. *Information Systems Research*, 2013.

[136] Kathy Chen Lionel Briand, Yvan Labiche. A Multi-Objective Genetic Algorithm to Rank State-Based Test Cases. 2013.

[137] Haikun Liu, Cheng-Zhong Xu, Hai Jin, Jiayu Gong, and Xiaofei Liao. Performance and Energy Modeling for Live Migration of Virtual Machines. In *Proceedings of the 20th International Symposium on High Performance Distributed Computing*, 2011.

[138] Liang Liu, Hao Wang, Xue Liu, Xing Jin, Wen Bo He, Qing Bo Wang, and Ying Chen. GreenCloud: A New Architecture for Green Data Center. In *Proceedings of the 6th International Conference Industry Session on Autonomic Computing and Communications Industry Session*, 2009.

[139] X. Liu, X. Zhu, S. Singhal, and M. Arlitt. Adaptive entitlement control of resource containers on shared servers. 2005.

[140] Zhenhua Liu, Yuan Chen, Cullen Bash, Adam Wierman, Daniel Gmach, Zhikui Wang, Manish Marwah, and Chris Hyser. Renewable and Cooling Aware Workload Management for Sustainable Data Centers. In *Proceedings of the*

12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems, 2012.

[141] Microsoft Office Live.

[142] Jochen Ludewig. Models in software engineering an introduction. *Software and Systems Modeling*, 2003.

[143] Sean Luke. Essentials of Metaheuristics, 2009.

[144] Michael Maurer, Ivona Brandic, and Rizos Sakellariou. Enacting SLAs in Clouds Using Rules. In *Euro-Par (1)*, 2011.

[145] Phil McMinn. Search-based Software Test Data Generation: A Survey: Research Articles. *Softw. Test. Verif. Reliab.*, 2004.

[146] David Meisner, Brian T Gold, and Thomas F Wenisch. PowerNap : Eliminating Server Idle Power. *Analysis*, 2009.

[147] Peter Mell and Timothy Grance. The NIST Definition of Cloud Computing. Technical report, National Institute of Standards and Technology (NIST), 2011.

[148] Stephen Milborrow. *Multivariate Adaptive Regression Spline Models*, 2007.

[149] Aleksandar Milenkoski, Alexandru Iosup, Samuel Kounev, Kai Sachs, Piotr Rygielski, Jason Ding, Walfredo Cirne, and Florian Rosenberg. Cloud Usage Patterns: A Formalism for Description of Cloud Usage Scenarios. Technical report, SPEC Research Group - Cloud Working Group, 2013.

[150] Theophano Mitsa. *Temporal Data Mining*. Chapman & Hall/CRC, 2010.

[151] Aziz Murtazaev and Sangyoon Oh. Sercon: Server Consolidation Algorithm using Live Migration of Virtual Machines for Green Computing. *IETE Technical Review*, 2011.

[152] Ripal Nathuji and Karsten Schwan. VirtualPower: Coordinated Power Management in Virtualized Enterprise Systems. *SIGOPS*, 2007.

[153] Qais Noorshams, Dominik Bruhn, Samuel Kounev, and Ralf Reussner. Predictive Performance Modeling of Virtualized Storage Systems Using Optimized Statistical Regression Techniques. In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, 2013.

[154] Qais Noorshams, Axel Busch, Samuel Kounev, and Ralf Reussner. The Storage Performance Analyzer: Measuring, Monitoring, and Modeling of I/O Performance in Virtualized Environments (Invited Demonstration Paper). In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, 2015.

[155] Qais Noorshams, Roland Reeb, Andreas Rentschler, Samuel Kounev, and Ralf Reussner. Enriching software architecture models with statistical models for performance prediction in modern storage environments. In *Proceedings of the 17th International ACM Sigsoft Symposium on Component-based Software Engineering*, New York, NY, US, 2014.

[156] omg. *Meta Object Facility (MOF) Core Specification Version 2.0*, 2006.

[157] Yew-Soon Ong, Zongzhao Zhou, and Dudy Lim. Curse and Blessing of Uncertainty in Evolutionary Algorithm Using Approximation. In *IEEE International Conference on Evolutionary Computation*, 2006.

[158] D. Pandit, S. Chattopadhyay, M. Chattopadhyay, and N. Chaki. Resource allocation in cloud using simulated annealing. In *Applications and Innovations in Mobile Computing (AIMoC), 2014*, 2014.

[159] C. Papagianni, A. Leivadeas, S. Papavassiliou, V. Maglaris, C. Cervello-Pastor, and A. Monje. On the optimal allocation of virtual resources in cloud computing networks. *Computers, IEEE Transactions on*.

[160] Roy P. Pargas, Mary Jean Harrold, and Robert R. Peck. Test-Data Generation Using Genetic Algorithms. *Software Testing, Verification And Reliability*, 1999.

[161] Dean F. Hougen Pedro A. Diaz-Gomez. Initial Population for Genetic Algorithms: A Metric Approach. 2007.

[162] ENERGY STAR Program. Report to Congress on Server and Data Center Energy Efficiency Public Law 109-431. Technical report, U.S. Environmental Protection Agency, 2007.

[163] Omer Rana, Martijn Warnier, Thomas B. Quillinan, Frances Brazier, and Dana Cojocarasu. Managing Violations in Service Level Agreements.

[164] Jia Rao, Xiangping Bu, Cheng Z. Xu, Leyi Wang, and George Yin. VCONF: A Reinforcement Learning Approach to Virtual Machines Auto-configuration. In *Proceedings of the 6th International Conference on Autonomic Computing*, 2009.

[165] Carl Edward Rasmussen. Gaussian processes for machine learning. MIT Press, 2006.

[166] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis. In *Proceedings of the Third ACM Symposium on Cloud Computing*, 2012.

[167] Luis Rodero-Merino, Luis M. Vaquero, Victor Gil, Fermín Galán, Javier Fontán, Rubén S. Montero, and Ignacio M. Llorente. From Infrastructure Delivery to Service Management in Clouds. *Future Gener. Comput. Syst.*, 2010.

[168] Robert Rose. Survey of system virtualization techniques. Technical report, 2004.

[169] M. Rosenblum and T. Garfinkel. Virtual machine monitors: current technology and future trends. *Computer*, (5), May 2005.

[170] J. W. Ross and G. Westerman. Preparing for utility computing: The role of IT architecture and relationship management. *IBM Syst. J.*, 2004.

[171] Franz Rothlauf. *Represenatations for Evolutionary and Genetic Algorithms*. 2006.

[172] AWS Autoscaling rules, http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide properties-as-policy.html.

[173] DW Morgan RV Krejcie. Determining sample size for research activities. 1970.

[174] T. Setzer and A. Stage. Decision support for virtual machine reassignments in enterprise data centers. In *Network Operations and Management Symposium Workshops (NOMS Wksps), 2010 IEEE/IFIP*, 2010.

[175] Robert H. Shumway and David S. Stoffer. *Time Series Analysis and Its Applications: With R Examples (Springer Texts in Statistics)*. Springer, 2006.

[176] D. Simon. Biogeography-Based Optimization. *Evolutionary Computation, IEEE Transactions on*, 2008.

[177] James E. Smith and Ravi Nair. The Architecture of Virtual Machines. *Computer*, 2005.

[178] Alex J. Smola and Bernhard Schölkopf. A Tutorial on Support Vector Regression. *Statistics and Computing*, 2004.

[179] Murat Kulahci Soren Bisgaard. *Time Series Analysis and Forecasting by Example*. Wiley, 2011.

[180] James Staten. Is The IaaS/PaaS Line Beginning To Blur?, 2011.

[181] Christopher Stewart and Kai Shen. Performance Modeling and System Management for Multi-component Online Services. In *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, 2005.

[182] T. Stützle. *Local Search Algorithms for Combinatorial Problems - Analysis, Algorithms and New Applications*. PhD thesis, TU Darmstadt, 1999.

[183] Brian Ripley Terry Therneau, Beth Atkinson. *Recursive Partitioning and Regression Trees*, 2014.

[184] Patrick Thibodeau. Data centers, under strain, expand at furious pace, May 2011.

[185] A. Tiwari, K. Vergidis, and B. Majeed. Evolutionary Multi-objective Optimization of Business Processes. In *IEEE CEC*, 2006.

[186] Jóakim v. Kistowski, Hansfried Block, John Beckett, Klaus-Dieter Lange, Jeremy A. Arnold, and Samuel Kounev. Analysis of the influences on server power consumption and energy efficiency for cpu-intensive workloads. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, 2015.

[187] Hien Nguyen Van, Frederic Dang Tran, and Jean-Marc Menaud. SLA-Aware Virtual Resource Management for Cloud Infrastructures. *2009 Ninth IEEE International Conference on Computer and Information Technology*, 2009.

[188] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A Break in the Clouds: Towards a Cloud Definition. *SIGCOMM*, 2008.

[189] A. Vasan, Anand Sivasubramaniam, V. Shimpi, T. Sivabalan, and R. Subbiah. Worth their watts? - an empirical study of datacenter servers. In *High Performance Computer Architecture (HPCA)*, 2010.

[190] David A. Van Veldhuizen and G.B. Lamont. On Measuring Multiobjective Evolutionary Algorithm Performance. In *In 2000 Congress on Evolutionary Computation*, 2000.

[191] Salvatore Venticinque, Rocco Aversa, Beniamino Di Martino, Massimilano Rak, and Dana Petcu. A Cloud Agency for SLA Negotiation and Management. In *Proceedings of the 2010 Conference on Parallel Processing*, 2011.

[192] Akshat Verma, Puneet Ahuja, and Anindya Neogi. pMapper: power and migration cost aware application placement in virtualized systems. In *Middleware '08: Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, 2008.

[193] Akshat Verma, Puneet Ahuja, and Anindya Neogi. Power-aware Dynamic Placement of HPC Applications. In *Proceedings of the 22Nd Annual International Conference on Supercomputing*, 2008.

[194] Akshat Verma, Gautam Kumar, and Ricardo Koller. The Cost of Reconfiguration in a Cloud. In *Proceedings of the 11th International Middleware Conference Industrial Track*, 2010.

[195] Akshat Verma, Gautam Kumar, Ricardo Koller, and Aritra Sen. CosMig: Modeling the Impact of Reconfiguration in a Cloud. In *MASCOTS*, 2011.

[196] William Voorsluys, James Broberg, Srikumar Venugopal, and Rajkumar Buyya. Cost of Virtual Machine Live Migration in Clouds: A Performance Evaluation. In *Proceedings of the 1st International Conference on Cloud Computing*, 2009.

[197] Chien-Hung Wei and Ying Lee. Sequential forecast of incident duration using Artificial Neural Network models. 2007.

[198] J. Wilkes and C. Reiss. Details of the ClusterData 2011-1 trace. 2011.

[199] James Williams. *A Novel Representation for Search-Based Model-Driven Engineering*. PhD thesis, University of York, 2013.

[200] Timothy Wood, Ludmila Cherkasova, Kivanc Ozonat, and Prashant Shenoy. Profiling and Modeling Resource Usage of Virtualized Applications. In *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, 2008.

[201] Timothy Wood, Prashant Shenoy, Arun Venkataramani, and Mazin Yousif. Black-box and gray-box strategies for virtual machine migration. In *Proceedings of the 4th USENIX conference on Networked systems design and implementation*, NSDI'07, pages 17–17, Berkeley, CA, USA, 2007. USENIX Association.

[202] LinLin Wu, Saurabh Kumar Garg, Steve Versteeg, and Rajkumar Buyya. SLA-based Resource Provisioning for Hosted Software as a Service Applications in Cloud Computing Environments. *IEEE Transactions on Services Computing*, 2013.

[203] Bo Xu, Zhiping Peng, Fangxiong Xiao, AntonioMarcel Gates, and Jian-Ping Yu. Dynamic deployment of virtual machines in cloud computing using multi-objective optimization. *Soft Computing*, 2014.

[204] Jing Xu and Jose A. B. Fortes. Multi-Objective Virtual Machine Placement in Virtualized Data Center Environments. In *Proceedings of the 2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing*, 2010.

[205] Li Xu, Zhibin Zeng, and Xiucai Ye. Multi-objective Optimization Based Virtual Resource Allocation Strategy for Cloud Computing. In *Proceedings of the 2012 IEEE/ACIS 11th International Conference on Computer and Information Science*, 2012.

[206] Taro Yamane. Statistics: An Introductory Analysis. 1967.

[207] Yagiz Onat Yazir, Chris Matthews, Roozbeh Farahbod, Stephen Neville, Adel Guitouni, Sudhakar Ganti, and Yvonne Coady. Dynamic Resource Allocation in Computing Clouds Using Distributed Multiple Criteria Decision Analysis. In *Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing*, 2010.

[208] Gioqinang Zhang and Michael Y. Hu. Neural network forecasting of the British Pound/US Dollar exchange rate. 1998.

[209] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 2010.

[210] Qi Zhang, Eren Gürses, Raouf Boutaba, and Jin Xiao. Dynamic Resource Allocation for Spot Markets in Clouds. In *Proceedings of the 11th USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, 2011.

[211] Qi Zhang, Mohamed Faten Zhani, Shuo Zhang, Quanyan Zhu, Raouf Boutaba, and Joseph L. Hellerstein. Dynamic Energy-aware Capacity Provisioning for Cloud Computing Environments. In *Proceedings of the 9th International Conference on Autonomic Computing*, 2012.

[212] He Zhao, Chenglei Peng, Yao Yu, Yu Zhou, Ziqiang Wang, and Sidan Du. Cost-Aware Automatic Virtual Machine Scaling in Fine Granularity for Cloud Applications. In *Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*, 2013.

[213] Donald W. Zimmerman. Invalidation of Parametric and Nonparametric Statistical Tests by Concurrent Violation of Two Assumptions. *The Journal of Experimental Education*, 1998.

[214] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. *IEEE Transactions on Evolutionary Computation*, 1999.

[215] Eckart Zitzler, Marco Laumanns, and Stefan Bleuler. A Tutorial on Evolutionary Multiobjective Optimization. In *In Metaheuristics for Multiobjective Optimisation*, 2003.

[216] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. Technical report, 2001.

[217] Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M. Fonseca, and Viviane Grunert da Fonseca. Performance Assessment of Multiobjective Optimizers: An Analysis and Review. *IEEE Transactions on Evolutionary Computation*, 7, 2002.